Grado en Ingeniería Informática
2016-2017

Trabajo Fin de Grado

ESTUDIO DE TÉCNICAS SUPERVISADAS DE REDUCCIÓN DE DIMENSIONALIDAD PARA PROBLEMAS DE CLASIFICACIÓN

Álvaro Soriano Maganto

Tutor: José María Valls Ferrán

Co-tutor: Ricardo Aler Mur

Colmenarejo, Junio de 2017

Resumen

En la actualidad es muy frecuente encontrar conjuntos de datos con una dimensión muy elevada. Por ello, en este trabajo se lleva a cabo el estudio de las diferentes técnicas existentes de reducción de dimensionalidad, así como los inconvenientes que presentan los algoritmos de aprendizaje automático en problemas con estas características.

Posteriormente, se lleva a cabo el estudio empírico de una técnica de reducción de dimensionalidad para problemas supervisados de clasificación. Esta técnica está basada en la arquitectura del Perceptrón Multicapa y el estudio es realizado en cuatro dominios diferentes. Adicionalmente, para comprobar si se trata de una técnica fiable, se muestra una comparación de los resultados obtenidos con PCA. Cabe destacar que para evaluar la calidad de la reducción de la dimensionalidad producida, en ambos casos, se utiliza KNN.

En la mayoría de los casos los resultados obtenidos realizando la reducción de la dimensionalidad con la red de neuronas son mejores que los resultados de PCA. También, en general se obtienen mejores resultados de clasificación de los datos con dimensiones reducidas (utilizando la red de neuronas para realizar la reducción de dimensionalidad) frente a los datos con el número de dimensiones original. Por tanto, el resultado del estudio es que es muy recomendable utilizar el esquema de reducción de dimensionalidad propuesto para llevar a cabo la transformación de los datos originales.

Palabras clave: Reducción de la dimensionalidad, selección de características, extracción de características, Redes de neuronas artificiales, Perceptrón Multicapa, PCA, KNN, Aprendizaje automático.

Índice

Índice de ilustraciones

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1. Introducción

En la actualidad se pueden encontrar múltiples problemas donde la dimensionalidad de los datos es muy elevada y esta tendencia está al alza. Este tipo de problemas suponen un desafío en diferentes áreas de investigación, en este caso todos los esfuerzos se centrarán en reducir el número de dimensiones de los datos para mejorar la eficiencia de los algoritmos de aprendizaje automático. Realizar esta tarea en muchos dominios es altamente recomendable porque dichos algoritmos escalan muy mal a medida que aumentan las dimensiones de los datos y reducen drásticamente su eficiencia en la resolución de problemas con estas características.

Una de las técnicas más ampliamente utilizada y conocida para la reducción de la dimensionalidad es Principal Component Analysis (PCA). Esta técnica lleva a cabo la transformación de los datos generando unas nuevas componentes, conocidas como *componentes principales*, que se calculan teniendo en cuenta la varianza del conjunto de datos. Sin embargo, presenta el principal problema que es una técnica no supervisada, es decir, no utiliza el valor que se espera obtener para cada dato. Por este motivo, en algunos dominios, PCA puede proporcionar resultados no satisfactorios.

Para solucionar los problemas que tiene PCA y producir una mejor transformación de los datos a un conjunto con menores dimensiones en problemas supervisados, se propone un esquema basado en redes de neuronas artificiales, concretamente utilizando el modelo que recibe el nombre de Perceptrón Multicapa (MLP). Este modelo está formado por neuronas artificiales que se organizan en diferentes tipos de capas.

En el Perceptrón Multicapa, la primera capa, también llamada *capa de entrada*, contiene neuronas que se encargan de distribuir los datos de entrada a las neuronas de la siguiente capa que recibe el nombre de *capa oculta*. Las neuronas de la capa oculta calculan su activación aplicando una función no lineal a la información que reciben de la capa de entrada y, a su vez, lo propagan a las neuronas de la siguiente capa. La última capa de la red, conocida como *capa de salida*, contiene las neuronas que dan una respuesta al exterior.

Podemos considerar que la capa oculta realiza una transformación del espacio de entrada de los datos originales con n dimensiones (tantas como neuronas de entrada) a un espacio diferente con m dimensiones (tantas como neuronas ocultas). Si m es menor que n, se habrá producido una reducción de dimensionalidad. Esta transformación de n a m dimensiones podrá ser lineal o no lineal: los valores que reciben las neuronas ocultas procedentes de la capa de entrada corresponden a una transformación lineal. Las neuronas ocultas calcularán sus activaciones aplicando a estos valores una función no lineal. Estas activaciones corresponden a la transformación no lineal de m dimensiones.

Teniendo en cuenta el razonamiento anterior, la arquitectura de la red neuronal propuesta para realizar la reducción de la dimensionalidad estará formada por una capa de entrada, una capa oculta y una capa de salida. En la capa de entrada se utilizarán tantas neuronas como dimensiones tenga el conjunto de datos original y la reducción de la dimensionalidad se producirá ya que, una vez realizado el entrenamiento de la red, se utilizarán las activaciones de las neuronas de la capa oculta como los atributos del nuevo conjunto de datos. Por tanto, el número de neuronas de la capa de entrada deberá ser estrictamente mayor que el número de neuronas de la capa oculta para poder llevar a cabo la reducción de forma apropiada. Cabe destacar que en la fase de entrenamiento será utilizada una única neurona en la capa de salida que proporcionará el valor esperado para cada patrón.

Adicionalmente, se llevará a cabo una comparación entre ambas técnicas para comprobar si se mejoran los resultados que aporta PCA en problemas supervisados de clasificación. Para poder realizar la comparación y evaluar la calidad de cada solución se utilizará un algoritmo de aprendizaje automático conocido con el nombre de K-nearest neighbors (KNN).

1.2. Objetivos

El objetivo principal de este proyecto es proponer un método de reducción de dimensionalidad supervisado basado en redes de neuronas artificiales y llevar a cabo un estudio experimental exhaustivo para evaluar dicho método con diferentes alternativas y parámetros.

En concreto, los objetivos específicos que se proponen para el presente trabajo son los siguientes:

- Realizar un estudio del estado del arte de los métodos existentes de reducción de la dimensionalidad.
- Implementación de una red de neuronas artificiales utilizando el Perceptrón Multicapa para problemas de clasificación biclase. La red deberá ser entrenada y será necesaria la implementación de un método para la obtención de las activaciones de las neuronas de la capa oculta.
- Implementación de PCA.
- Estudio empírico de la reducción de la dimensionalidad producida por la capa oculta del Perceptrón Multicapa. En dicho estudio se comprobará si, clasificando los datos con dimensiones reducidas, se mejoran los resultados de clasificación de los datos originales. Adicionalmente, se realizará una comparación con los resultados proporcionados por PCA.
- Si los resultados obtenidos en la experimentación son buenos, se creará una librería con el método de reducción de dimensionalidad estudiado. Dicha librería podrá ser utilizada de forma libre por la comunidad y será integrada en *Caret*.

1.3. Definiciones, abreviaturas y acrónimos

1.3.1. Definiciones

Algoritmo: Es un conjunto ordenado y finito de reglas o instrucciones que tienen el propósito de buscar la solución a un problema mediante sucesivos pasos.

Array: También conocido como *vector* o *arreglo*. Se trata de una estructura de almacenamiento que sirve para almacenar datos de forma consecutiva en memoria.

Capacidad de generalización: Es la capacidad que tiene un modelo generado con un algoritmo de aprendizaje automático para responder de forma adecuada ante nuevos patrones de datos que no han sido utilizados en el proceso de entrenamiento.

Data frame: Es una estructura de almacenamiento presente en algunos lenguajes de programación. Se trata de una lista de arrays de igual tamaño y sirve para almacenar datos en tablas.

Dominio: También conocido como *conjunto de datos* o *base de datos*. En aprendizaje automático, es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

Función: En programación, una función es un grupo de instrucciones con un objetivo en particular y que se ejecuta al ser llamada desde otra función. Las funciones pueden recibir datos a través de los *parámetros* y deben retornar o devolver un resultado.

Librería: En informática, una librería es un conjunto de subprogramas implementados para facilitar la tarea de desarrollo de otro tipo de programas más complejos.

Script: Es un programa almacenado en un fichero de texto plano. Consisten en una serie de órdenes utilizadas para automatizar tareas.

Semilla: En informática, es un valor inicial utilizado por un generador de números aleatorios para generar números de forma pseudoaleatoria. Siempre que se parte de la misma semilla se obtiene la misma secuencia de valores.

Software: Es un conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

1.3.2. Acrónimos y siglas

Acrónimo	Significado
MLP	Multi-layer Perceptron (Perceptrón Multicapa)
PCA	Principal Component Analysis
S	Sigmoidal
TL	Transformación Lineal
TLN	Transformación Lineal Normalizada
KNN	K-Nearest Neighbors
RNA	Redes de Neuronas Artificiales

Tabla 1.1: *Acrónimos*

1.4. Estructura del documento

El presente documento se extiende a lo largo de ocho capítulos y un anexo. A continuación, se detallan los aspectos que se tratan en cada capítulo.

En el *capítulo 1*, llamado *Introducción y objetivos*, se realiza una introducción del trabajo llevado a cabo, se describen los objetivos que se persiguen y se muestra la estructura general del documento. También se muestran definiciones, abreviaturas y acrónimos.

En el *capítulo 2*, llamado *Estado del arte*, se realiza un planteamiento del problema, indicando las ventajas que aporta la reducción de la dimensionalidad y los efectos negativos que producen los dominios con un número elevado de dimensiones sobre los algoritmos de aprendizaje automático. Además, se presentan aplicaciones reales y trabajos similares. En este capítulo también se realiza un estudio del estado del arte de las técnicas de reducción de dimensionalidad, así como del resto de conceptos y técnicas que se emplean a lo largo del desarrollo del trabajo. Cabe destacar que también se comparan las soluciones presentadas.

En el *capítulo 3*, llamado *Marco regulador*, se realiza un estudio de la ley vigente aplicable al ámbito del trabajo desarrollado. También se comenta el estándar técnico que se ha seguido en la extracción de requisitos. Además de esto, se calculan los rangos salariales para los cargos desempeñados en el trabajo.

En el *capítulo 4*, llamado *Entorno socio-económico*, se muestra la planificación inicial estimada del proyecto, se realiza un presupuesto de la elaboración del trabajo y, por último, se lleva a cabo un estudio del impacto socio-económico.

En el *capítulo 5*, llamado *Diseño de la solución*, se realiza una explicación de la metodología empleada para llevar a cabo la experimentación. Además, se presentan las alternativas de diseño que se tuvieron en cuenta durante el desarrollo del trabajo. Por último, en este capítulo también se muestran los requisitos del software desarrollado y se explica en detalle el código implementado.

En el *capítulo 6*, llamado *Estudio realizado*, se describen los dominios que son utilizados para realizar el estudio empírico del esquema propuesto y se muestran todos los resultados obtenidos en los diversos experimentos. Adicionalmente, se realiza un análisis de los resultados.

En el *capítulo 7*, llamado *Conclusiones*, se muestran las principales conclusiones obtenidas como resultado del desarrollo del trabajo. En este apartado también se muestran algunas de las líneas de investigación que quedan planteadas como trabajo futuro.

En el *anexo I*, llamado *Abstract*, se realiza un resumen en inglés de las principales partes del trabajo.

Capítulo 2

Estado del arte

2.1. Preprocesado de datos

El preprocesado de datos es el proceso en el cual se lleva a cabo la manipulación de un conjunto de datos. En muchos casos es necesario aplicar este tipo de técnicas a los datos antes de poder ser utilizados por algoritmos de aprendizaje automático.

En la actualidad existen multitud de técnicas de preprocesamiento de datos, cabe destacar que las técnicas de reducción de dimensionalidad son consideradas como una forma de preprocesado de datos. En este apartado se describirán también otros métodos de preprocesado de datos que son utilizados a lo largo del trabajo. Dichas técnicas son la *numerización*, *normalización* y la *aleatorización* de los datos.

2.1.1. Numerización

En primer lugar, clasificaremos los atributos como nominales o numéricos. Los atributos numéricos son aquellos que únicamente toman valores reales o enteros, mientras que los nominales son atributos discretos o categóricos.

Como veremos en futuros apartados, los algoritmos que se van a utilizar no son capaces de trabajar con atributos nominales, por lo que es necesario realizar lo que se conoce como una *numerización*. Esto se consigue asignando a cada valor discreto un número entero o real.

2.1.2. Normalización

Una vez que tenemos todos nuestros atributos son de tipo numérico, se puede realizar una normalización de los datos. Esto quiere decir que los datos son expresados en un determinado intervalo, generalmente en el intervalo [0,1]. Para realizar la normalización es necesario aplicar la siguiente fórmula a cada instancia:

$$VariableNormalizada_i = \frac{Variable_i - Mínimo_i}{Máximo_i - Mínimo_i} \quad (2.1)$$

Donde:

- $Variable_i$ corresponde con la variable i
- $VariableNormalizada_i$ es la variable i expresada en el intervalo [0,1]
- $Mínimo_i$ corresponde con el valor mínimo del atributo i para todas las instancias.
- $Máximo_i$ corresponde con el valor máximo del atributo i para todas las instancias

2.1.3. Aleatorización

Para evitar sesgos en la distribución de los datos es conveniente hacer una aleatorización de los datos. Después de realizar esto, el conjunto de datos se separa en varios conjuntos de datos:

- *Conjunto de datos de entrenamiento*: Este conjunto de datos sirve para realizar el entrenamiento del algoritmo.
- *Conjunto de datos de validación*: Este conjunto de datos es opcional y se utiliza para comprobar cómo evoluciona el proceso de entrenamiento o para decidir los mejores parámetros para el modelo.
- *Conjunto de datos de test*: Este conjunto de datos sirve para evaluar lo aprendido por el algoritmo.

2.1.4. Reducción de la dimensionalidad

La dimensión de los datos es el número de *variables*, también conocido como *atributos* o *características*, que son medidas en cada observación. Por tanto, la reducción de la dimensionalidad, sobre un conjunto de datos con un número elevado de variables, hace referencia al proceso de reducción del número de dimensiones de los datos originales generando un conjunto de datos con una capacidad predictiva similar, pero con menor número de variables.

El objetivo general de la reducción de la dimensionalidad de un conjunto de datos es eliminar variables que sean irrelevantes o redundantes. Sin embargo, realizando esto podemos perder gran cantidad de información valiosa para resolver el problema. Por este motivo, la clave consiste en tratar de reducir la dimensionalidad sin sacrificar la información más importante del conjunto de datos.

2.1.4.1. Técnicas para la reducción de la dimensionalidad

Las técnicas de reducción de dimensionalidad pueden ser divididas en dos grandes grupos: los *métodos de selección de características* y los *métodos de extracción de características*.

2.1.4.1.1. Métodos de selección de características

Los métodos de selección de características consisten en elegir un subconjunto de los atributos originales de los datos. En este caso no se realiza ninguna transformación de los datos originales, solamente se seleccionan aquellos atributos que se consideran más importantes. Por tanto, un atributo es seleccionado si se considera relevante para la resolución del problema o descartado porque se considera irrelevante.

Dentro de los métodos de selección de características existen tres estrategias distintas: *Filtrado*, *envolventes* e *incrustados*.

Métodos envolventes

En esta estrategia se intenta elegir el subconjunto de atributos que proporcione el mejor resultado posible con un algoritmo de aprendizaje concreto.

Los métodos envolventes utilizan algoritmos de búsqueda para buscar en todo el espacio de posibles subconjuntos de atributos. Estos subconjuntos se evalúan generando

un modelo mediante un algoritmo de aprendizaje automático. Por tanto, consiste en utilizar el rendimiento de un modelo para evaluar la utilidad de un subconjunto de atributos. Teniendo en cuenta esto, para utilizar este método es necesario definir:

- Estrategia de búsqueda: Se puede realizar búsqueda en escalada, mejor-primero, etc. Además, es posible comenzar por el conjunto de atributos de entrada completo o el conjunto vacío de atributos.
- Algoritmo de aprendizaje automático para generar el modelo: El método para generar el modelo durante la fase de selección de atributos no es necesario que sea el método que se utilizará finalmente. Habitualmente para intentar evitar el sobreajuste se suelen utilizar algoritmos que sean robustos y cuyos parámetros sean fáciles de ajustar, como puede ser *random forest*.
- Método de evaluación del modelo generado.

Como se puede suponer, este método tiene el principal inconveniente de que es muy costoso computacionalmente. Sin embargo, proporciona el mejor o uno de los mejores subconjuntos de atributos para un tipo de modelo concreto.

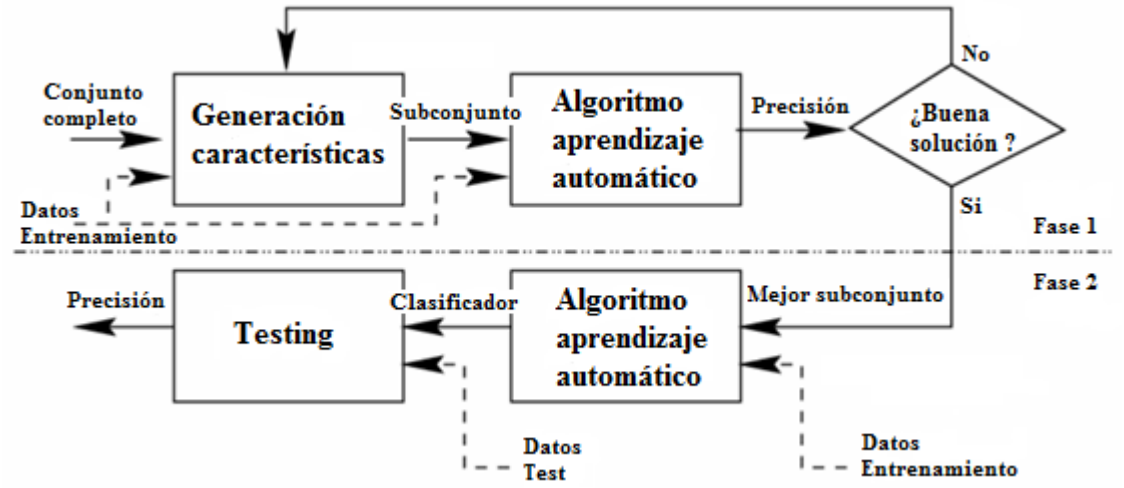

Ilustración 2.1: *Esquema de funcionamiento de los métodos envolventes*

Un ejemplo de método envolvente es el algoritmo conocido con el nombre de *Boruta* que utiliza *random forest* para evaluar la calidad de un subconjunto de atributos concreto. Su funcionamiento consiste en cada iteración añadir atributos aleatorios, también llamados como atributos *sombra*, que son obtenidos a partir de los atributos originales. Posteriormente, se crea un modelo utilizando *random forest* y se calcula la importancia relativa de cada variable. Si se detecta que una variable original está por debajo de una variable sombra, ésta es eliminada del modelo. Este proceso se repite hasta que todas las variables se aceptan, se rechazan o se alcanza un número concreto de iteraciones.

Métodos de filtrado

En esta estrategia se utilizan criterios para la selección de los atributos que son independientes del algoritmo con el que se realiza el ajuste del modelo y, por tanto, proveen una selección genérica de atributos que es indiferente del algoritmo utilizado. Por ejemplo, un criterio para la selección puede ser medir la tasa de correlación entre cada atributo y la clase, o realizar un test de contraste de hipótesis.

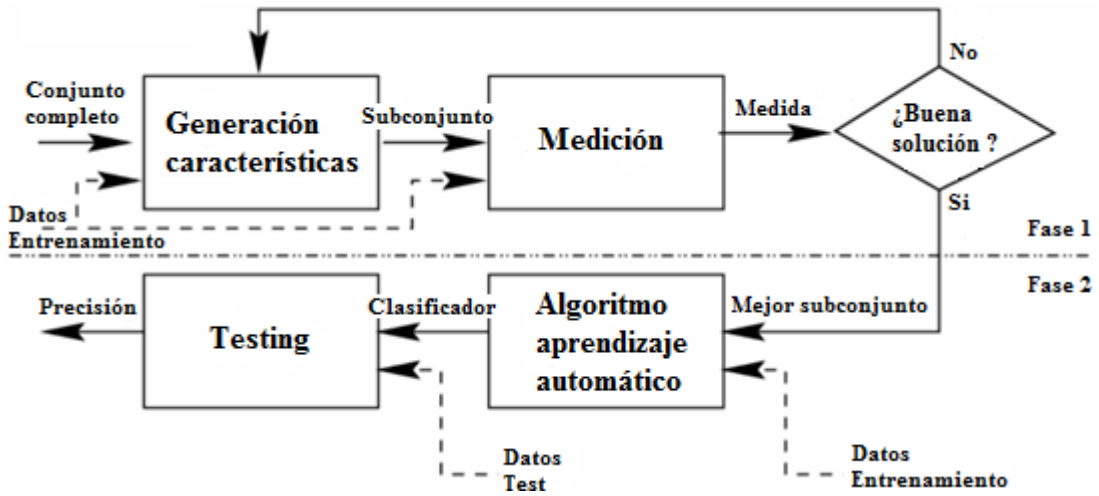

Ilustración 2.2: *Esquema de funcionamiento de los métodos de filtrado*

La mayor ventaja de este tipo de métodos es que son rápidos comparados con los métodos envolventes y escalan mejor a medida que aumenta el número de atributos, ya que solo necesita el cálculo de medidas simples. Sin embargo, su principal inconveniente es que tienden a eliminar atributos que estén poco correlacionados con la clase, pero que con la interacción con otra variable pueda producir buenos resultados.

Como se demuestra en [Guyon et al., 2005], para un mejor rendimiento en los métodos de selección de características se pueden combinar los métodos de filtrado y los envolventes. En primer lugar, se realiza un filtro para obtener un conjunto de datos con un número de atributos moderado y, posteriormente, se aplican métodos de filtrado para encontrar un subconjunto de atributos lo más adecuado posible.

Una técnica de filtrado es *Minimum-redundancy-maximum-relevance (mRMR)* que utiliza la correlación de cada variable con la clase para la selección de las variables y, además, penaliza la redundancia con otras variables.

Métodos incrustados

Los métodos incrustados han sido propuestos para combinar las ventajas de los dos métodos anteriores y son altamente especializados. En este tipo de técnicas, el algoritmo de aprendizaje es modificado para ser capaz de realizar una selección de atributos. Dicho algoritmo recibe un conjunto de datos y automáticamente genera un modelo con un menor número de atributos, es decir, no hay un paso concreto para la selección de atributos.

2.1.4.1.2. Métodos de extracción de características

Los métodos de extracción de características realizan una transformación completa de los datos originales de alta dimensión a un conjunto de menor dimensión tratando de retener la mayor cantidad de información posible.

Existen multitud de algoritmos de extracción de características y se pueden clasificar en técnicas supervisadas y no supervisadas. Las técnicas supervisadas son aquellas que realizan la transformación de los datos originales teniendo en cuenta la solución del problema. Por el contrario, las técnicas no supervisadas realizan la transformación solo teniendo en cuenta los datos.

Algunas de los métodos de extracción de características más empleados son PCA (véase *2.2. Principal Component Analysis (PCA)*) y autocodificadores (véase *2.4.2.5. Autocodificadores*). Es importante destacar que ambas técnicas se describen de forma detallada más adelante, ya que es necesario explicar previamente algunos conceptos adicionales.

2.1.4.2. Planteamiento del problema

A menudo puede parecer que un algoritmo de aprendizaje automático funcionará mejor cuando disponga de mucha información en cada instancia de entrenamiento, sin embargo esto en la práctica no funciona así. Los algoritmos empeoran su rendimiento a medida que aumentamos la dimensión de los datos.

En la siguiente ilustración se muestra como evoluciona de forma general la eficiencia de los algoritmos de aprendizaje automático con el aumento de las dimensiones. Se observa que se alcanza un máximo en la eficiencia con un número concreto de dimensiones y a medida que aumentamos las dimensiones el rendimiento empieza a decaer.

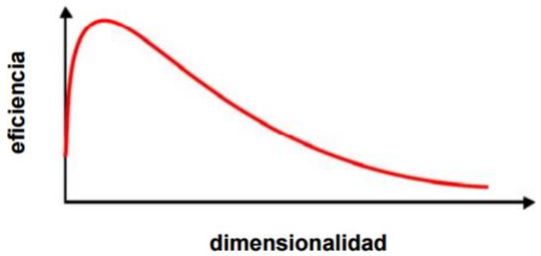

Ilustración 2.3: *Evolución de la eficiencia con el aumento de las dimensiones*

Por tanto, el principal motivo por el que se debe realizar una reducción de la dimensionalidad es para reducir el efecto conocido como *maldición de la dimensionalidad* (en inglés, *curse of dimensionality*). Este fenómeno fue descubierto por Richard E. Bellman [Bellman, 1961; Bellman y Corporation, 1957] cuando trabajaba con problemas de optimización dinámica. Consiste en que a medida que aumentamos el número de dimensiones, el volumen del espacio crece exponencialmente y provocamos que los datos estén más dispersos, lo que produce que la cantidad de datos de entrenamiento necesarios para extraer resultados fiables crezca también exponencialmente.

La maldición de la dimensionalidad puede ser interpretada visualmente como un espacio casi vacío donde es complicado encontrar patrones en los datos de ejemplo para tomar decisiones. Esta situación provoca que los modelos obtenidos se ajusten a particularidades de los datos de entrenamiento.

En la *ilustración 2.4* podemos apreciar gráficamente que es lo que ocurre a medida que aumentamos las dimensiones de los datos. En la imagen en una dimensión (imagen de la izquierda) podemos observar que solo hay 10 regiones de interés, por lo que un algoritmo con suficientes ejemplos de entrenamiento dentro de estas regiones podría generalizar de forma correcta. En la imagen en dos dimensiones (imagen del centro) el número de regiones aumenta a 100 regiones (10x10), esto provoca la necesidad de tener ejemplos de entrenamiento en todas las regiones. Por último, en la imagen en tres dimensiones el número de regiones crece a 1.000 regiones (10x10x10). Esta situación se

puede generalizar que para d dimensiones y v valores se necesitan al menos $O(v^d)$ regiones y ejemplos [Goodfellow et al., 2016].

Ilustración 2.4: *Maldición de la dimensionalidad*

Adicionalmente, en muchos dominios es posible la reducción de la dimensionalidad porque muchos atributos están altamente correlacionados con otros o simplemente son irrelevantes para la resolución del problema, es decir, son solo ruido. Estos atributos además de no contribuir nada al proceso de aprendizaje pueden incluso degradarlo por el fenómeno explicado con anterioridad.

Un dominio con un número elevado de atributos también influye negativamente en los algoritmos de aprendizaje automático, ya que produce grandes retardos en el proceso de entrenamiento. Además de esto, provoca problemas de sobreajuste (el concepto de sobreajuste se explica con detalle en el apartado *2.4.2.3. Expresividad. Subajuste y sobreajuste*) de los datos de entrenamiento debido a que aumentan los grados de libertad del sistema.

Otro motivo por el que históricamente se han aplicado las técnicas de reducción de la dimensionalidad ha sido para poder visualizar los datos de alta dimensión realizando una proyección en 2 o 3 dimensiones.

2.1.4.3. Aplicaciones y trabajos similares

Para el desarrollo del trabajo se han tenido en cuenta los trabajos similares que se incluyen a continuación:

- En [Segovia, 2015] se lleva a cabo un estudio de reducción de dimensionalidad utilizando una red de neuronas para problemas supervisados. En este caso se realiza un estudio empírico, utilizando KNN para comprobar el rendimiento de la reducción de la dimensionalidad en cinco dominios distintos. Además, también se realiza una comparación de los resultados obtenidos por la red de neuronas con PCA.
- En [Guyon et al., 2005] se muestra el análisis de los resultados obtenidos en una competición organizada para comprobar el rendimiento de la reducción de la dimensionalidad utilizando métodos de selección de características. En esta competición se realizó un estudio empírico de las distintas técnicas planteadas empleando cinco dominios distintos.
- En [Wack et al., 2006] se realiza un estudio del rendimiento de tres algoritmos de reducción de dimensionalidad aplicados a datos musicales. Cabe destacar que uno de los algoritmos estudiados es PCA.

Por otro lado, algunas de las aplicaciones más importantes de la reducción de la dimensionalidad son las siguientes:

- En la categorización de textos dentro de categorías, como se realiza en [Yang y Pederson, 1997], donde cada atributo corresponde a la ocurrencia de una palabra clave o un término clave del documento.
- En [Swets y Weng, 1995; Dy et al., 2003] se aplica en la recuperación de imágenes, donde se ha producido una explosión en la colección de imágenes y es necesario indexar las imágenes para recuperarlas de forma eficiente.
- En reconocimiento de caras como es llevado a cabo en [Shylaja, Balasubramanya y Natarajan, 2011] y en el reconocimiento de dígitos manuscritos como podemos comprobar en [LeCun et al., 1990], donde en ambos casos cada atributo corresponde con un píxel de la imagen. Incluso una imagen pequeña de 64 píxeles por 64 píxeles tendría más de 4.000 atributos lo que hace que las imágenes sean muy difíciles de manejar.
- La reducción de la dimensionalidad es aplicada para la detección de intrusiones en sistemas informáticos en [Lee et al., 2000]. En este caso se analiza el tráfico usando algoritmos de aprendizaje automático y se determina si existe un intruso o no.
- En [Golub et al., 1999] se aplica la reducción de la dimensionalidad y después se lleva a cabo la clasificación de nuevas muestras en tipos de enfermedad conocidos. En este caso cada atributo corresponde con un gen. Puede haber cientos de genes y pocos ejemplos.
- En imágenes hiperespectrales como podemos apreciar en [Lodha y Kampalur, 2014]. En este tipo de imágenes se generan conjuntos de datos de altas dimensiones, ya que cada píxel puede contener más de 200 medidas espectrales en diferentes longitudes de onda.
- En la clasificación de los componentes químicos guardados por un espectrómetro de masas. Las características de un componte químico puede ser representado por cientos de características.

2.2. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) es la técnica estadística más tradicional y conocida que existe, fue propuesta por Karl Pearson [Pearson, 1901] y más tarde desarrollada por Harold Hotelling [Hotelling, 1933]. Esta técnica se trata de un método de reducción de la dimensionalidad mediante extracción de características.

El método consiste en la transformación de las variables de entrada, probablemente relacionadas, a otro conjunto de variables llamadas *componentes principales* que son independientes. Estas nuevas componentes, ortogonales y linealmente independientes entre sí, se obtienen mediante la combinación lineal de las variables originales.

$$Y_1 = a_{11}X_1 + a_{12}X_2 + \cdots + a_{1d}X_d$$
$$Y_2 = a_{21}X_1 + a_{22}X_2 + \cdots + a_{2d}X_d$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$
$$Y_d = a_{d1}X_1 + a_{d2}X_2 + \cdots + a_{dd}X_d$$

Ilustración 2.5: *Transformación lineal realizada por PCA*

Donde:

- Y_d es el *d-ésimo* componente principal.
- X_d es el *d-ésimo* componente original.

Desde el punto de vista geométrico, PCA puede ser visto como una rotación de los ejes originales sobre sus medias.

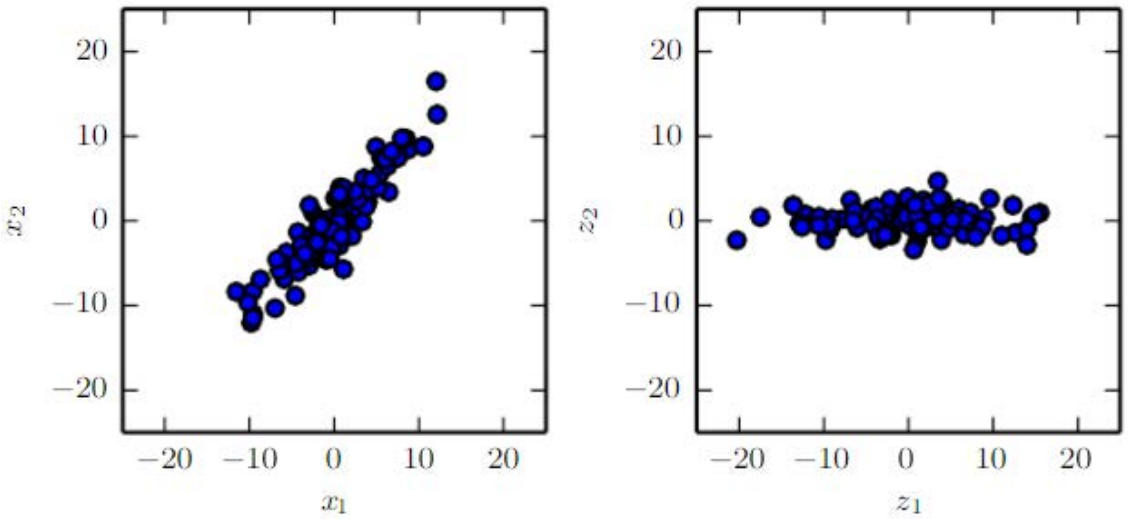

Ilustración 2.6: *Rotación de los datos.*

El primer componente principal es definido por el método donde el conjunto de datos presenta una mayor varianza, el segundo componente principal es definido donde los datos presentan la segunda mayor varianza, y así sucesivamente. Hay que tener en cuenta que el número máximo de componentes principales que se obtienen mediante PCA siempre será igual al número de variables del conjunto de datos, aunque como el objetivo es reducir la dimensionalidad habrá que reducir el número de componentes seleccionando aquellos que expliquen la mayor cantidad de variabilidad de los datos.

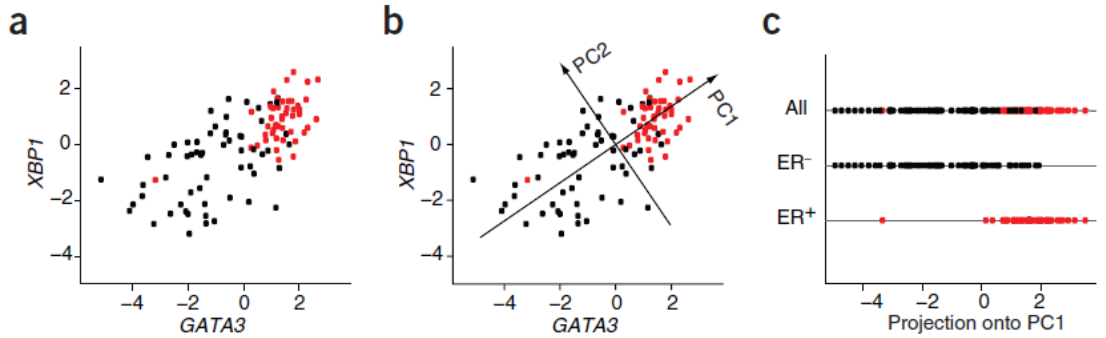

Ilustración 2.7: *Ejemplo de funcionamiento de PCA*

En la anterior ilustración (*Ilustración 2.7: Ejemplo de funcionamiento de PCA*) podemos visualizar un ejemplo extraído de [Ringnér, 2008] en forma gráfica. En este ejemplo los datos están etiquetados con dos clases distintas (ER+ y ER-), podemos visualizarlo en la figura (a) como puntos rojos (ER+) y negros (ER-). En la figura (b) vemos los componentes principales (PC1 y PC2) sobre el conjunto de datos. En la figura (c) se muestran los datos sobre una única dimensión, es decir, se ha eliminado la segunda componente (PC2); se puede apreciar que se ha perdido información quitando un componente, pero es posible extraer conclusiones visualizando como los datos de las dos clases están bastante separados, es decir, en extremos opuestos de la gráfica. Nótese que si los datos son proyectados sobre el PC2 las conclusiones que se podrían obtener serían engañosas, ya que veríamos los datos de las clases juntos.

Por tanto, para este método es importante ordenar las componentes principales de acuerdo a la varianza de los datos, siendo el componente principal más importante el que más variabilidad explica de los datos y el componente menos importante el que menos varianza tiene. De acuerdo a esta reflexión, para reducir la dimensionalidad de los datos, eliminaremos las variables que menos cantidad de variabilidad explican.

Esta técnica presenta como principal ventaja que retiene las características del conjunto de datos que contribuyen más a su varianza. Sin embargo, tiene el inconveniente que no ofrece resultados satisfactorios en muchos problemas supervisados de clasificación o regresión al tratarse de una técnica no supervisada.

Hay que tener en cuenta que existen distintas formas de realizar el cálculo de las componentes principales. En este caso se va a explicar detalladamente el proceso más conocido que utiliza la descomposición ortogonal de la matriz de covarianzas. Sin embargo, como se demuestra en [Shlens, 2003], es posible utilizar la descomposición de valores singulares (SVD) de la matriz de datos y, en la actualidad, es el método más eficiente para el cálculo de PCA.

2.2.1. Cálculo de los componentes principales

La covarianza entre dos variables mide el nivel de relación que existe entre ellas. La fórmula para calcular esta medida es la siguiente:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{(n-1)} \tag{2.2}$$

La matriz de covarianzas recoge las covarianzas entre cada par de variables de un determinado conjunto de datos. Esta matriz siempre es simétrica, ya que se cumple que $cov(X,Y) = cov(Y,X)$. Además, la diagonal principal contiene las varianzas de cada variable porque se cumple que $cov(X,X) = var(X)$.

Por tanto, para construir la matriz de covarianzas de un conjunto de datos con *p* variables es necesario calcular la covarianza entre cada par de variables y ordenarlas de la siguiente forma:

$$S = \begin{pmatrix} cov(X,X) & cov(X,Y) & \cdots & cov(X,P) \\ cov(Y,X) & cov(Y,Y) & \cdots & cov(Y,P) \\ \vdots & \vdots & & \vdots \\ cov(P,X) & cov(P,Y) & \cdots & cov(P,P) \end{pmatrix} \tag{2.3}$$

La matriz de covarianzas se puede construir de forma equivalente a la anterior realizando el producto matricial que se puede ver en la ecuación (2.4). Es importante destacar que la matriz *B* es la matriz de datos centrados en media 0, esto se consigue restando la media de cada dimensión a toda la columna de datos.

$$S = \frac{1}{n-1} BB^T \tag{2.4}$$

Por tanto, el método PCA se basa en la diagonalización ortogonal de una matriz *S* que corresponde con la matriz de covarianza del conjunto de datos. La diagonalización ortogonal consiste en encontrar una matriz diagonal *L* semejante a *S*, de la siguiente forma:

$$U^T SU = L \tag{2.5}$$

Donde los vectores columna de la matriz *U* son los autovectores ortonormales de la matriz *S*, mientras que los elementos de la diagonal de la matriz *L* son los autovalores de la matriz *S.* Además, como se demuestra en [Hernando, 2013], se cumple que toda matriz simétrica es ortogonalmente diagonalizable y, por tanto, la matriz de covarianzas al ser simétrica siempre será diagonalizable ortogonalmente.

El primer paso del método para calcular los componentes principales consiste en obtener la matriz de covarianzas de los datos, utilizando la ecuación (2.3) o la (2.4).

Una vez calculada la matriz, es necesario su descomposición en autovalores (eigenvalor) y autovectores (eigenvector). Para realizar esto, es necesario resolver la *ecuación característica*:

$$|S - \lambda I| = 0 \tag{2.6}$$

Donde:

- *S* es la matriz de covarianza.
- *I* es la matriz identidad.
- λ son la autovalores.

Cuando se desarrolla la ecuación característica se obtiene un polinomio en función de λ que es conocido como *polinomio característico*. Resolviendo esta ecuación obtenemos los distintos autovalores λ. Estos valores indican la cantidad de información que tiene cada componente, cuanto mayor sea el autovalor λ significará que más cantidad de varianza explica su autovector y, por ende, más importante será.

Para calcular los autovectores es necesario resolver, para cada autovalor obtenido, el sistema matricial: $S - \lambda I = 0$.

En [Hernando, 2013] se demuestra que para toda matriz simétrica se cumple que los autovectores asociados a autovalores distintos son ortogonales entre sí. Teniendo en cuenta esto, como la matriz *S* (matriz de covarianzas) es simétrica, los autovectores asociados a distintos autovalores obtenidos serán ortogonales. Sin embargo, los autovectores asociados al mismo autovalor pueden no ser ortogonales, en este caso será necesario ortogonalizarlos mediante algún método, como por ejemplo el proceso de Gram-Schmidt [Afken, 1985].

Como se comentó anteriormente, la matriz *U*, que contiene los autovectores, debe ser ortogonal, esto quiere decir que sus vectores columna deben ser ortonormales. Por ello, se deben normalizar los autovectores $\vec{v}$. Para realizar esto, se divide cada autovector por su módulo:

$$\vec{u} = \frac{\vec{v}}{|\vec{v}|} \tag{2.7}$$

Es importante destacar que los ejes de coordenadas de los componentes principales están definidos por los autovectores.

2.2.2. Transformación de los datos originales

Una vez que tenemos los autovectores, estos deben ser ordenados por su correspondientes autovalores asociados, de mayor a menor. Este orden da los componentes por orden de significancia y es posible formar una matriz *U*:

$$U = (eig_1\ eig_2 \dots\ eig_n) \tag{2.8}$$

Donde eig_1 tiene el mayor autovalor y corresponde con el primer componente (PC1), eig_2 tiene el segundo mayor autovalor y corresponde con el segundo componente (PC2), y así sucesivamente.

Para reducir la dimensionalidad de los datos, es posible construir la matriz *U*, eligiendo los mejores *k* autovectores del conjunto total de *n* autovectores calculados, siendo $n > k$, perdiendo así la mínima cantidad de información posible en el nuevo conjunto de datos.

$$U = (eig_1\ eig_2 \dots\ eig_k) \tag{2.9}$$

Existen diferentes criterios para elegir con cuántas dimensiones quedarse en cada caso. Uno de los más conocidos consiste en elegir los atributos que expliquen al menos un 75% de la varianza de los datos. Otro criterio consiste en representar gráficamente los autovalores en un gráfico denominado *scree diagram* y elegir los atributos donde la pendiente cambia más bruscamente.

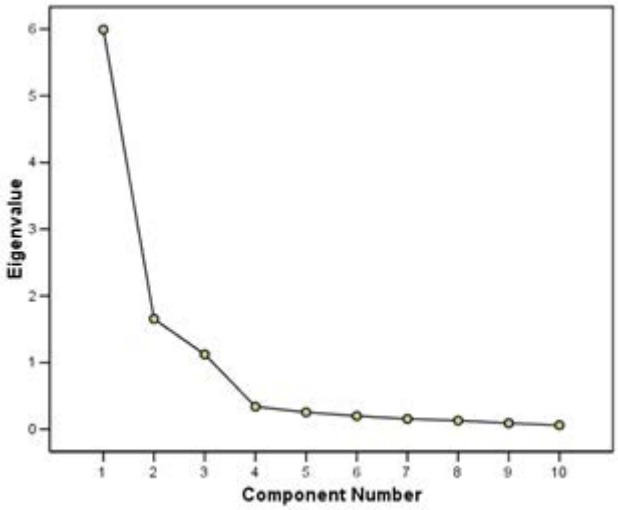

Ilustración 2.8: *Scree diagram*

Por último, se aplica la transformación lineal de la ecuación (2.10) a cada uno de los datos.

$$\vec{y} = U[\vec{x} - \bar{x}] \tag{2.10}$$

En este caso el sistema de ecuaciones lineales del anterior apartado (*Ilustración 2.5: Transformación lineal realizada por PCA*) se ha expresado en forma matricial, pero el resultado es el mismo. Además, es importante destacar que, como se comentó anteriormente, cada dato debe estar centrado con media cero y no sobre los datos originales, por este motivo se realiza la resta $[\vec{x} - \bar{x}]$ antes de realizar la multiplicación con la matriz *U*.

Cada dato individual transformado es conocido comúnmente con el nombre de *score*, mientras que los coeficientes de la matriz *U* son ampliamente conocidos con el nombre de *loading*.

2.3. Aprendizaje automático

El aprendizaje automático es un campo de la Inteligencia Artificial que permite a las máquinas la capacidad de aprender. Esto elimina la necesidad de haber sido programados específicamente para resolver un problema concreto.

Estos sistemas basados en ejemplos utilizan un conjunto de datos para aprender y se someten a un proceso de entrenamiento, etapa en la cual el algoritmo debe ser capaz de extraer las características más relevantes de los datos de entrenamiento para, posteriormente, poder generalizar de forma apropiada. Por tanto, se trata de una inducción del conocimiento.

La fase de entrenamiento es muy importante en este tipo de algoritmos y para ser realizado de forma satisfactoria el conjunto de datos de entrenamiento debe cumplir los siguientes requisitos [Isasi y Galván, 2003]:

- Ser significativo: El conjunto de datos debe ser lo suficientemente grande.
- Ser representativo: El conjunto de datos tiene que tener variedad suficiente en los datos. Si disponemos de unos datos desbalanceados corremos el riesgo de que el algoritmo se especialice en un subconjunto y no tendrá buena capacidad de generalización.

2.3.1. Tareas del aprendizaje automático

En la práctica existen diferentes tipos de tareas y cada una de ellas tiene sus propias características que son las que definen su tipo y cómo debe ser resuelto. De forma general estas tareas pueden ser clasificadas como *tareas predictivas* y *descriptivas*. Por tanto, el aprendizaje automático tiene que ser capaz de resolver distintos tipos de tareas.

2.3.1.1. Tareas predictivas

Se caracterizan fundamentalmente porque cada instancia de entrenamiento contiene la solución al problema. Dentro de las tareas predictivas se puede destacar la *regresión* y la *clasificación*.

2.3.1.1.1. Regresión

Un problema es de regresión cuando necesitamos obtener como solución un valor continuo. De forma matemática se puede definir como la necesidad de encontrar una función *f* tal que $f: \mathbb{R}^n \rightarrow \mathbb{R}$.

Un ejemplo de problema de regresión puede ser la predicción de la nota númerica que un alumno puede sacar en una determinada asignatura en función de sus datos personales, como puede ser la edad, gustos musicales, etc.

Medidas de evaluación para una tarea de regresión

Una vez generado el modelo de regresión por un algoritmo de aprendizaje automático, es necesario evaluarlo. Para realizar esto, es habitual utilizar medidas como son el error medio o el error medio cuadrático sobre el conjunto de test.

$$ErrorMedio = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{Y}_i - Y_i\right| \quad (2.11)$$

$$ErrorCuadraticoMedio = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2 \quad (2.12)$$

Donde:

- $\hat{Y}_i$ es la salida obtenida por el modelo para el patrón *i*
- Y_i es la salida esperada para el patrón *i*

2.3.1.1.2. Clasificación

Un problema es de clasificación cuando la solución se trata de un conjunto de valores discretos definidos previamente. De forma matemática puede ser definido como la necesidad de encontrar una función *f* tal que $f: \mathbb{R}^n \rightarrow \{1, \dots, k\}$.

Si el número total de clases es de exactamente dos, se conoce como clasificación binaria o biclase. Por el contrario, si el número de clases es superior a dos, se conoce con el nombre de clasificación multiclase.

Un ejemplo de clasificación podría ser un sistema de concesión de crédito, las clases en este caso serían dos: el crédito es concedido y el crédito no es concedido. Los atributos serían los datos de la persona que solicita el crédito, como podría ser su sueldo, edad, etc.

Es importante destacar, ya que se utilizará en el desarrollo del presente proyecto, que todo problema de clasificación puede ser convertido en un problema de regresión y viceversa, por lo que es posible resolver problemas de clasificación con algoritmos de regresión modificando el planteamiento del problema.

Medidas de evaluación para una tarea de clasificación

Para evaluar el modelo de clasificación generado por un algoritmo de aprendizaje automático se utiliza el porcentaje de acierto o el porcentaje de error que se comete en el conjunto de test.

$$PorcentajeAcierto = \frac{a}{n} \quad (2.13)$$

$$PorcentajeError = \frac{f}{n} \quad (2.14)$$

Donde:

- *a* es el número total de aciertos
- *f* es el número total de fallos
- *n* es el número total del conjunto de datos

Hay que tener en cuenta que estos resultados pueden ser engañosos, ya que por ejemplo en un hipotético dominio biclase donde el porcentaje de aparición de la primera clase es del 90% y la segunda clase solo se da en el 10% de los casos, si el modelo generado clasifica todas las entradas en la primera clase, se obtendría un 90% de acierto que es un porcentaje muy elevado y, sin embargo, este modelo sería inútil. Por tanto, para analizar más profundamente el resultado de un modelo se suele utilizar la matriz de confusión.

		Resultado obtenido	
		Positivo	Negativo
Resultado esperado	Positivo	Verdadero positivo (TP)	Falso negativo (FN)
	Negativo	Falso positivo (FP)	Verdadero negativo (TN)

Tabla 2.1: *Matriz de confusión*

Utilizando la matriz de confusión, se pueden obtener las siguientes tasas que proporcionan mucha información acerca del modelo generado:

- El porcentaje de aciertos total: $PR = \frac{TP+TN}{TP+TN+FN+FP}$
- El porcentaje de aciertos de la clase positivo: $TPR = \frac{TP}{TP+FN}$
- El porcentaje de aciertos de la clase negativo: $TNR = \frac{TN}{FP+TN}$

2.3.1.2. Tareas descriptivas

Se caracterizan porque solo se contienen los datos de entrenamiento y, por tanto, no contienen la solución del problema. En este grupo se puede destacar las técnicas de *agrupación* y las *reglas de asociación*.

2.3.1.2.1. Agrupación

Un problema es de agrupación cuando tratamos de buscar patrones de comportamiento similares en los datos, es decir, organizar los datos en grupos que tengan características comunes. En la siguiente ilustración podemos observar un ejemplo gráfico, a la izquierda se observa el conjunto de datos original y a la derecha los mismos datos con su agrupación más natural.

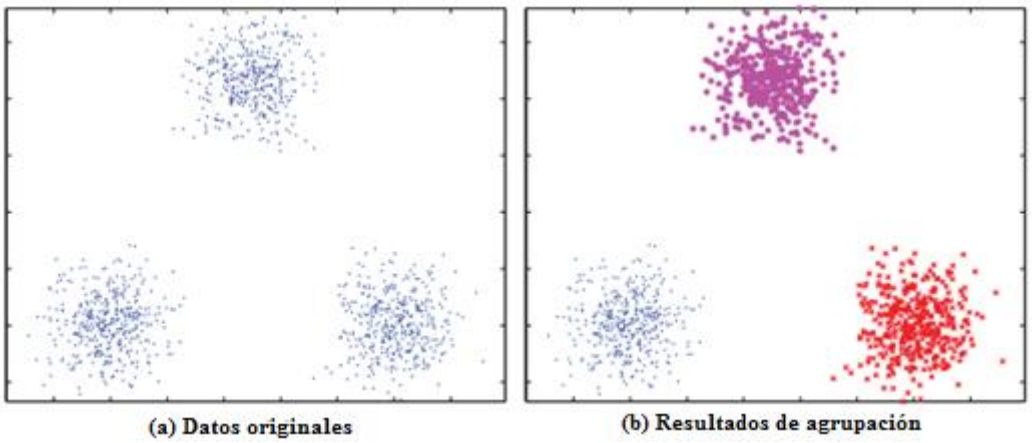

Ilustración 2.9: *Ejemplo de problema de agrupación*

Matemáticamente se puede definir de forma idéntica a la clasificación, como la necesidad de encontrar una función *f* tal que $f: \mathbb{R}^n \rightarrow \{1, \dots, k\}$. Sin embargo, ambos se diferencian en que en la agrupación a priori no se conoce cómo son los grupos ni cuántos hay.

Un ejemplo de este tipo de problema puede ser encontrar grupos de clientes con comportamientos similares a partir de la base de datos de las compras que han realizado.

Medidas de evaluación para una tarea de agrupación

Para medir la calidad de la agrupación, aunque depende del algoritmo utilizado, es común utilizar la cohesión y la separación entre grupos. Para calcular estas medidas, se suele utilizar la distancia media de todos los puntos a su centro y la distancia media entre grupos, respectivamente. La distancia utilizada generalmente corresponde con la distancia euclídea.

2.3.1.2.2. Reglas de asociación

El análisis de asociaciones se utiliza para encontrar dependencias o relaciones entre los atributos. En este caso solo se utilizan atributos nominales y se caracteriza porque, a partir del conjunto de ejemplos $A_1 \times A_2 \times \dots \times A_n$, se encuentran reglas de asociación tal que "$si\ A_i = a \wedge A_j \wedge \dots \wedge A_k = h\ entonces\ A_r = u \wedge A_s = v \wedge \dots \wedge A_z = w$".

Un ejemplo muy típico de este tipo de tarea es el análisis de la cesta de la compra, donde se quieren conocer qué productos se han comprado juntos, de tal forma que los productos relacionados estén situados lo más próximo posible en los estantes.

Regla 1: $Cereales \rightarrow Leche$
Regla 2: $Leche \wedge jamón \rightarrow queso \wedge pan$
Regla 3: $Leche \wedge pan \rightarrow queso \wedge jamón$
Regla 5: $Queso \wedge jamón \rightarrow leche \wedge pan$
Regla 6: $Queso \wedge pan \rightarrow leche \wedge jamón$
Regla 7: $Jamón \wedge pan \rightarrow leche \wedge queso$
Regla 8: $Jamón \rightarrow leche, queso \wedge pan$
Regla 9: $Pan \rightarrow leche \wedge queso \wedge jamón$

Ilustración 2.10: *Ejemplo de reglas de asociación*

Medidas de evaluación para una tarea de análisis de asociación

El algoritmo más conocido para realizar un análisis de asociación es *A priori* [Agrawal y Srikant, 1994], al cual se le debe proporcionar el umbral de cobertura que corresponde con el porcentaje mínimo de ejemplos en los que aparecen el subconjunto. Para generar las reglas de asociación también se debe utilizar la confianza que es la proporción de instancias que la regla predice correctamente.

2.3.2. Tipos de algoritmos de aprendizaje automático

Existen diferentes algoritmos de aprendizaje automático que dependen de las necesidades del problema a resolver. Los principales son los algoritmos con *aprendizaje supervisado* y *aprendizaje no supervisado*, aunque existen otras técnicas menos habituales como son el *aprendizaje por refuerzo* y el *aprendizaje semisupervisado*.

2.3.2.1. Aprendizaje supervisado

En el *aprendizaje supervisado* cada instancia de entrenamiento contiene la información necesaria del problema y la solución del mismo, es decir, se tratan de ejemplos etiquetados. En este tipo de algoritmos hay, por tanto, dos tipos de atributos distintos: los datos o información del problema y la solución del problema. Son generalmente utilizados para resolver problemas de clasificación o regresión.

2.3.2.2. Aprendizaje no supervisado

En este tipo de aprendizaje se contienen los datos del problema, pero no se tiene información acerca de la solución. Por tanto, en el *aprendizaje no supervisado* se busca encontrar grupos de ejemplos que sean lo más parecidos entre sí, y el aprendizaje suele ser guiado por la similitud de los ejemplos de entrenamiento. Cabe destacar que las tareas descriptivas son un tipo de aprendizaje no supervisado.

2.3.2.3. Aprendizaje semisupervisado

El etiquetado de los datos de entrenamiento es llevado a cabo por un humano y suele ser una tarea que requiere mucho tiempo cuando el volumen de datos es muy grande. Por el contrario, la adquisición de datos sin etiquetar es generalmente fácil. Por ello, el *aprendizaje semisupervisado* consiste en etiquetar una pequeña parte de las instancias de entrenamiento.

Un ejemplo de este tipo de aprendizaje es el *co-entrenamiento* que consiste en entrenar dos o más clasificadores, cada uno con un conjunto de datos independientes con diferentes atributos. Una vez que se tienen varios modelos ajustados, se etiquetan las demás instancias con la clase que diga el clasificador con más confianza.

2.3.2.4. Aprendizaje por refuerzo

La idea del *aprendizaje por refuerzo* es favorecer los comportamientos de forma que se maximice la recompensa o premio acumulado. En este tipo de aprendizaje, el sistema lleva a cabo una autoexploración del entorno aplicando una serie de acciones mediante un proceso iterativo en forma de prueba y error. Por ello, el objetivo es ir mejorando en la selección de las decisiones en el futuro, en función de las recompensas obtenidas.

2.3.3. Técnicas de evaluación

Como ya se comentó anteriormente, los algoritmos de aprendizaje automático suelen utilizar al menos dos conjuntos de datos diferentes. Por un lado, el primero de ellos, que es conocido como conjunto de entrenamiento, es utilizado en el proceso de aprendizaje para construir el modelo y, por otro lado, el segundo conjunto de datos es conocido como conjunto de test y es utilizado para medir la precisión del modelo. El conjunto de test no se utiliza nunca en el proceso de aprendizaje.

Una técnica de evaluación conocida con el nombre de *validación simple* consiste en separar previamente el conjunto de datos inicial en los dos conjuntos de datos necesarios de entrenamiento y test. Es importante destacar que para evitar sesgos en los datos se suele realizar una aleatorización de los datos antes de realizar la separación de los datos. El porcentaje de datos escogido para el conjunto de datos de test suele variar entre el 5% y el 50% del conjunto de datos total.

La técnica de *validación simple* presenta algunos problemas sobre todo en dominios donde no hay almacenada gran cantidad de ejemplos en los datos iniciales y no es posible reservar una parte de esos datos para test. Además de esto, también se puede dar la situación en la que después de haber realizado la aleatorización de los datos siga existiendo sesgo en los mismos. Para solucionar este tipo de problemas se suele utilizar otra técnica de evaluación conocida con el nombre *validación cruzada*.

La *validación cruzada con n particiones* consiste en separar los datos en *n* conjuntos distintos, donde el proceso de aprendizaje es llevado a cabo con *n-1* conjuntos de datos y se reserva un conjunto para evaluar el modelo. Dicho proceso de aprendizaje se realiza *n* veces, reservando cada vez un conjunto de datos distinto para test, por lo que obtendremos *n* tasas de error distintas. El último paso consiste en generar el modelo con todo el conjunto de datos disponible y se calcula la tasa de error media. Para esta técnica generalmente se utiliza un valor de *n* de 10.

Otra técnica de evaluación utilizada es *bootstrapping* que es especialmente indicada para casos en los que existen muy pocos ejemplos en los datos. En esta estrategia se generan distintos conjuntos de datos, conocidos con el nombre de *bootstrap samples*, donde se realiza un muestreo de los datos con reemplazamiento, por ello un mismo ejemplo de entrenamiento puede estar varias veces repetido. Sin embargo, existirán algunos datos que no estén en este conjunto de datos, ya que pertenecen al conjunto de test. El procedimiento es igual que en la validación cruzada, es decir, se realizan *n* entrenamientos del modelo, obteniendo *n* tasas de error distintas y finalmente se realiza el entrenamiento con todos los datos realizando un promedio de la tasa de error.

2.3.4. Métodos de aprendizaje automático

Existen diferentes tipos de tareas que debe resolver el aprendizaje automático, por lo que en la actualidad existen múltiples métodos. Cada uno de estos métodos está especializado en resolver una o varias tareas y funcionan mejor en unos dominios que en otros, dependiendo de sus características, no existiendo una regla universal para definir cuál es mejor en cada caso.

A continuación, se va a realizar una revisión de algunos de los métodos más destacados del aprendizaje automático, remarcando sus características principales.

Métodos bayesianos

En *métodos bayesianos* se suele utilizar la técnica conocida como *Naive Bayes* que está basada en la regla de Bayes. Es conocida con el nombre de ingenua (en inglés, *naive*) porque asume que los atributos son independientes.

Árboles de decisión

Los *árboles de decisión* generan decisiones secuenciales que se organizan de forma jerárquica, por este motivo reciben el nombre de árbol. Su objetivo es generar el árbol que mejor separe el conjunto de datos. Generalmente, utilizan la entropía para elegir los atributos que dividen el espacio de una forma más adecuada. Por tanto, emplean la estrategia "divide y vencerás" para partir el problema en subconjuntos.

Se pueden utilizar en tareas de clasificación, agrupamiento y regresión. Algunos de los algoritmos más conocidos de este método son ID3 y J48.

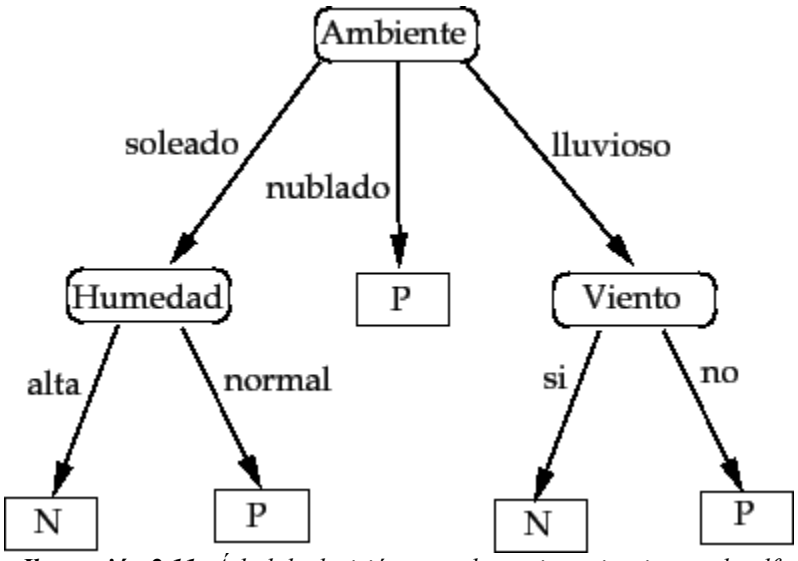

Ilustración 2.11: *Árbol de decisión para determinar si se juega al golf*

Una vez que se tiene generado el árbol, es posible generar fácilmente un conjunto de reglas del tipo "$Si\ cond_1 \wedge cond_2 \wedge \ldots \wedge cond_n\ entonces\ pred$", donde cada rama puede ser considerada como una regla. Los antecedentes de la regla son definidos por cada nodo interno, desde la raíz hasta las hojas, y el nodo hoja correspondiente define el consecuente de la regla. Así por ejemplo, las siguientes reglas son obtenidas a partir del árbol de la *ilustración 2.11*.

$$\boldsymbol{Si}\ Ambiente = soleado \wedge Humedad = alta\ \boldsymbol{entonces}\ jugar = N$$
$$\boldsymbol{Si}\ Ambiente = soleado \wedge Humedad = normal\ \boldsymbol{entonces}\ jugar = P$$
$$\boldsymbol{Si}\ Ambiente = nublado\ \boldsymbol{entonces}\ jugar = P$$
$$\boldsymbol{Si}\ Ambiente = lluvioso \wedge Viento = si\ \boldsymbol{entonces}\ jugar = N$$
$$\boldsymbol{Si}\ Ambiente = lluvioso \wedge Viento = no\ \boldsymbol{entonces}\ jugar = P$$

Ilustración 2.12: *Reglas construidas con el árbol de decisión de la ilustración 2.11*

Redes de neuronas artificiales

Las *redes de neuronas artificiales* son un paradigma muy potente bioinspirado, es decir, tratan de simular el sistema nervioso animal para resolver distintos tipos de problemas. Este paradigma puede ser utilizado en tareas de clasificación, regresión y agrupación. En el apartado *2.4. Redes de neuronas artificiales* se profundiza acerca de este tipo de métodos.

Aprendizaje basado en instancias

En el método conocido como *aprendizaje basado en instancias*, también conocido como *aprendizaje perezoso*, el objetivo es realizar poco esfuerzo a la hora de aprender, sin perjudicar la tarea de aprendizaje. Por tanto, el proceso de aprendizaje simplemente consiste en almacenar todos o algunos de los ejemplos de entrenamiento, por lo que en este caso no se generan estructuras de representación a partir de los datos de entrenamiento.

Puede ser utilizado en tareas de regresión, clasificación y agrupamiento. El algoritmo más conocido de este paradigma es KNN (*k-nearest neighbors*), que se explica en detalle en el apartado *2.6. K-nearest neighbors (KNN)*.

Algoritmos evolutivos

Los *algoritmos evolutivos* también son un paradigma bioinspirado que trata de imitar la evolución biológica. Son ampliamente utilizados en problemas de búsqueda donde el espacio de estados es demasiado grande y, aunque no aseguran encontrar la solución óptima en tiempo finito, suelen encontrar buenas soluciones. Se pueden usar en tareas de clasificación, agrupamiento y reglas de asociación.

Cada individuo o propuesta de solución se denomina *cromosoma.* Estos cromosomas están compuestos por *genes* que son utilizados para definir los rasgos del individuo. El número de *genes* que componen un *cromosoma* depende de la codificación del problema. Además, existe una *función de adaptación* que mide la calidad de los individuos, es utilizada para guiar el proceso de entrenamiento y también es dependiente del problema.

La población inicial está compuesta por una población de individuos que es generada frecuentemente al azar.

Aplicando *operadores genéticos* a los individuos seleccionados se crea una nueva generación de individuos. Este proceso se repite de forma iterativa durante múltiples generaciones hasta que se cumple una condición de parada, definida previamente.

```
Inicio Algoritmo Genético
      Iniciar población de individuos (al azar)
      Evaluar población
      Durante un número de generaciones o hasta que se cumpla
      una condición
            Seleccionar individuos para reproducción
            Reproducir (y mutar)
            Evaluar nuevos individuos
            Insertar nuevos individuos en la población
      Fin Durante
Fin Algoritmo Genético
```

Ilustración 2.13: *Esquema general de un algoritmo evolutivo*

Frecuentemente, los algoritmos evolutivos suelen utilizarse junto con otros métodos de aprendizaje automático, como puede ser con las redes de neuronas artificiales para ajustar los pesos de las conexiones de las neuronas.

2.4. Redes de neuronas artificiales

2.4.1. Introducción

El conocimiento de la base del funcionamiento de las neuronas biológicas motivó a diversos científicos a crear modelos computacionales basados en el funcionamiento de dichas neuronas. A esta técnica de Inteligencia Artificial se le conoce con el nombre de *redes de neuronas artificiales* (RNA).

En la actualidad existen múltiples modelos de redes de neuronas artificiales con distintas funciones de activación, reglas de aprendizaje y arquitecturas. En este apartado se va a explicar el modelo genérico de red de neurona artificial.

2.4.1.1. La neurona artificial

La neurona artificial trata de reproducir lo más fielmente posible el comportamiento de una neurona biológica. En un esquema artificial, cada neurona tiene *n* entradas que llamaremos $x_1, x_2, \dots, x_n$ y son las salidas de otras neuronas o los datos de entrada. Las conexiones tienen asignado un *peso* que denotaremos como $w_1, w_2, \dots, w_n$ y simulan la sinapsis. Si el peso entre dos neuronas conectadas es negativo, se produce un efecto de inhibición; mientras que si es positivo, se produce un efecto de excitación. Además de estos pesos, cada neurona incluye una entrada externa adicional, llamada *bias*, que se utiliza para aumentar o disminuir el nivel de activación de la neurona.

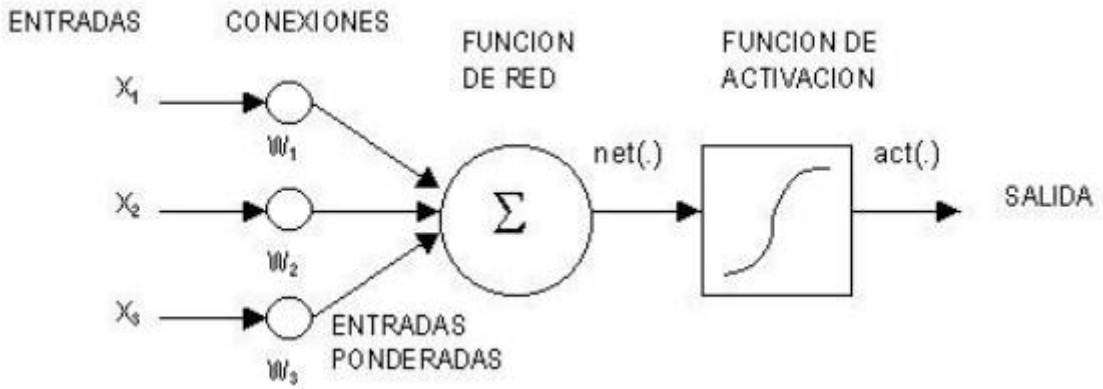

Ilustración 2.14: *Partes de una neurona artificial*

Hay que tener en cuenta que los valores de los *pesos* y el *bias* son calculados durante el proceso de entrenamiento. Generalmente, estos valores inicialmente tienen valores aleatorios y se van modificando de acuerdo a una ley de aprendizaje.

Cuando se recibe una entrada, la neurona realiza un sumatorio de cada entrada ponderado por su correspondiente peso asociado, como se realiza en la siguiente ecuación:

$$E = \sum_i w_i x_i + b \tag{2.15}$$

Esto puede ser definido de forma vectorial como sigue:

$$E = X^T W \tag{2.16}$$

Posteriormente, a la señal E se le aplica una función matemática f denominada *función de activación* que será la salida de la neurona. Existen distintas *funciones de activación* que se pueden aplicar, algunos ejemplos son los siguientes:

- Función lineal: $f(x) = \sum_i w_i x_i + b$
- Función umbral: $f(x) = \begin{cases} 0, si \ \sum_i w_i x_i + b \geq \theta \\ 1, si \ \sum_i w_i x_i + b < \theta \end{cases}$
- Función sigmoidal: $f(x) = \frac{1}{1+e^{-x}}$
- Función tangente hiperbólica: $f(x) = \frac{1-e^{-x}}{1+e^{-x}}$

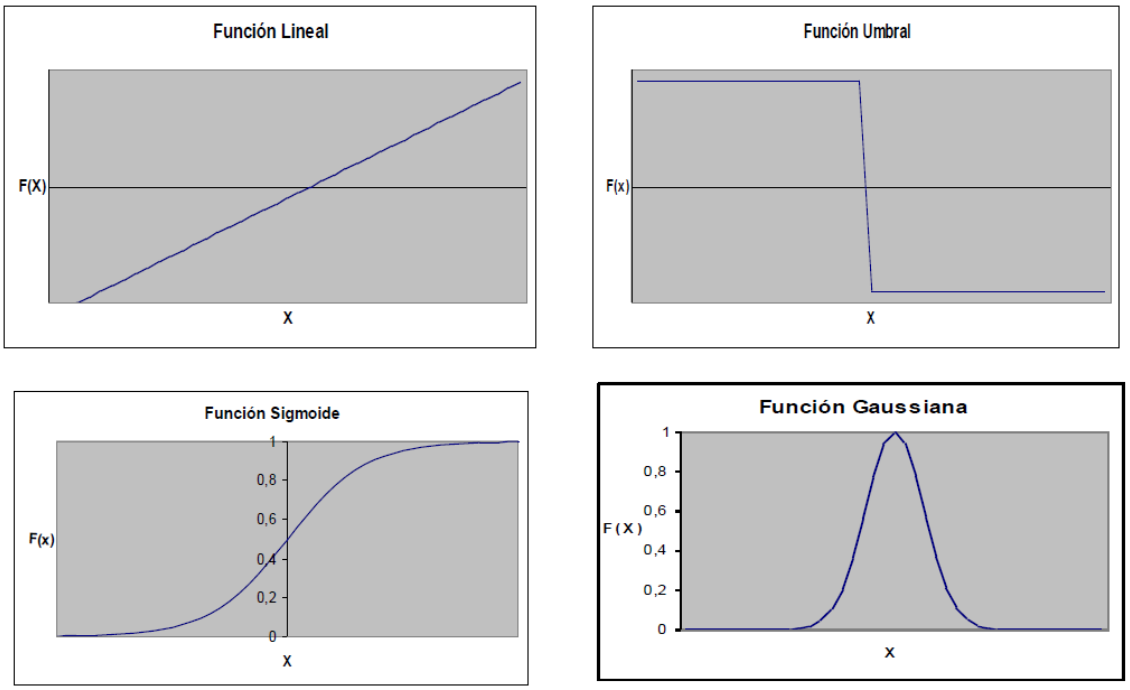

Ilustración 2.15: *Ejemplos de funciones de activación*

Una neurona artificial es una unidad de procesamiento muy simple y el verdadero potencial se obtiene cuando multitud de neuronas son conectadas entre sí formando redes que, generalmente, se suelen organizar por capas, aunque varía dependiendo de la estructura del modelo de red neuronal que adoptemos.

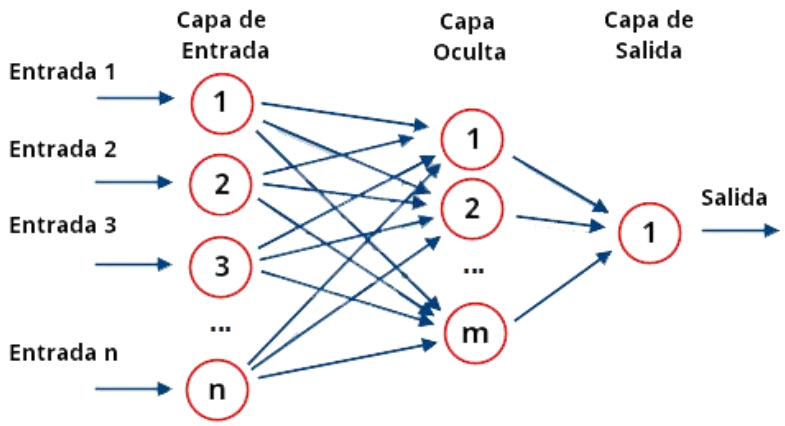

Ilustración 2.16: *Ejemplo de arquitectura de una red de neuronas artificial*

2.4.2. Perceptrón Multicapa (MLP)

En este apartado se va a describir el modelo conocido como Perceptrón Multicapa (MLP). El MLP es una generalización del Perceptrón Simple y surge motivado fundamentalmente por los problemas de separabilidad no lineal que presentaba este último.

Se trata de un modelo muy potente, ya que se trata de un aproximador universal. Esto quiere decir que cualquier función continua en $\mathbb{R}^n$ puede ser aproximada por un Perceptrón Multicapa con al menos una capa oculta.

2.4.2.1. Arquitectura del Perceptrón Multicapa

La arquitectura del Perceptrón Multicapa se organiza en capas de neuronas donde las conexiones son hacia delante, es decir, las salidas de las neuronas de la capa anterior corresponden con la entrada de las neuronas de la capa siguiente, por este motivo también reciben el nombre de *redes feedforward*. En este modelo existen tres tipos de capas:

- *Capa de entrada*: Solamente existe una capa de este tipo, las neuronas que la forman reciben la información del exterior y simplemente se encargan de distribuirla hacia las neuronas de la siguiente capa.
- *Capa oculta*: Existe una o varias capas ocultas. Cada capa oculta contendrá sus propias neuronas.
- *Capa de salida*: Existe una capa de salida y se encarga de proporcionar la respuesta al exterior.

En la *ilustración 2.17* podemos observar la arquitectura típica de un Perceptrón Multicapa. Como ya se comentó anteriormente, las conexiones entre cada par de neuronas tienen asociado un *peso* y cada neurona posee un *bias.*

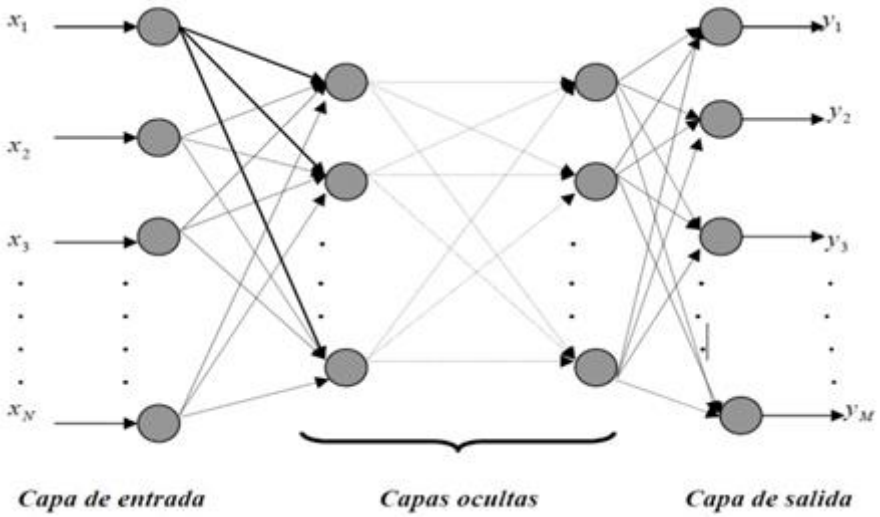

Ilustración 2.17: *Arquitectura típica del Perceptrón Multicapa*

Es importante destacar que las neuronas de la capa de entrada no realizan cómputo alguno y solo propagan los datos hacia delante. Por tanto, su nivel de activación se determina con la ecuación (2.17).

$$a_i^1 = x_i \text{ para } i = 1, 2, \ldots, k \qquad (2.17)$$

Sin embargo, las neuronas de las capas ocultas y la capa de salida siguen el funcionamiento de una neurona artificial típica (véase el apartado *2.4.1.1. La neurona artificial*) y su nivel de activación viene determinado por la siguiente ecuación.

$$a_i^c = f(\sum_{j=1}^{n_c-1} w_{ji}^{c-1} a_j^{c-1} + b_i^c) \text{ para } i = 1, 2,\ldots,n_c \text{ y } c = 2, 3,\ldots, \text{C-1} \qquad (2.18)$$

Es muy habitual encontrar la arquitectura de la *ilustración 2.17*, donde cada neurona de la capa anterior está conectada únicamente con todas las neuronas de la capa siguiente, es decir, la red está totalmente conectada. Sin embargo, existen otras dos aproximaciones:

- Las neuronas de la capa anterior solo son conectadas con algunas de las neuronas de la siguiente capa.
- Las neuronas de una capa se conectan con las neuronas de otra capa que no es inmediatamente posterior.

Diseño de la arquitectura del Perceptrón Multicapa

Un diseñador de una red de neuronas basada en el Perceptrón Multicapa debe tomar varias decisiones de diseño para abordar el problema. Dichas decisiones son las siguientes:

- La *función de activación* para las neuronas: En este caso se suele utilizar la función sigmoidal o la función tangente hiperbólica, la elección de una u otra no condiciona en absoluto a la solución del problema ya que ambas son similares. La elección se suele realizar dependiendo de la activación que se desea que tengan las neuronas. Generalmente, se suele utilizar la misma función de activación en todas las neuronas de la red, pero existen algunos casos en los que se utiliza una función de activación lineal o umbral en las neuronas de la capa de salida. Es importante destacar que no es conveniente utilizar una función de activación lineal en todas las neuronas de la red porque la composición de funciones lineales genera una función lineal.
- El número de neuronas de la capa de entrada y salida: La cantidad de neuronas de la capa de entrada viene determinado por el número de atributos y las neuronas de la capa salida también viene determinado por el problema, aunque en ocasiones un mismo problema se puede plantear de varias formas distintas y variar el número de neuronas de la capa de salida.
- El número de capas ocultas y el número de neuronas que tiene cada capa oculta: No está definido, ni existe ninguna regla para establecer estos parámetros, por lo que frecuentemente se realiza mediante ensayo y error.

2.4.2.2. Entrenamiento de Perceptrón Multicapa

El proceso de entrenamiento consiste en ajustar los parámetros de la red, es decir, los *pesos* de las conexiones y las *bias*. El algoritmo utilizado para llevar a cabo esta tarea en un Perceptrón Multicapa es el algoritmo de *Retropropagación*.

Como se trata de aprendizaje supervisado, el objetivo consiste en minimizar una función de error, *E*, que evalúa la diferencia entre la salida esperada y la obtenida por la red. Dicha función de error se suele definir de la siguiente forma:

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n) \qquad (2.19)$$

Siendo *N* es el número de ejemplos de entrenamiento y $e(n)$ es el error de la red para el ejemplo *n* del conjunto de entrenamiento. Generalmente, el error $e(n)$ que se suele utilizar es el error cuadrático medio, se describe matemáticamente de la siguiente forma:

$$e(n) = \frac{1}{2}\sum_{i=1}^{n_c}(s_i(n) - y_i(n))^2 \quad (2.20)$$

Donde:

- $y_i(n)$ es la salida *i* esperada para el ejemplo de entrenamiento *n*.
- $s_i(n)$ es la salida *i* deseada para el ejemplo de entrenamiento *n*.

Como se trata de un problema no lineal, es necesario utilizar una técnica de optimización no lineal sin restricciones para resolverlo, concretamente en el campo de las redes de neuronas artificiales se suele utilizar el método del *descenso del gradiente*. Dicha técnica consiste en variar los parámetros en función del sentido negativo del gradiente de la función *E*, es decir, la dirección por donde decrece más rápidamente dicha función. Por tanto, el funcionamiento del descenso del gradiente consiste minimizar el error para cada ejemplo entrenamiento, en lugar de minimizar el error global.

$$w(n) = w(n-1) - \alpha\frac{\partial e(n)}{\partial w} \quad (2.21)$$

Operando la ecuación (2.21) mediante la regla de la cadena, como se demuestra en [Isasi y Galván, 2003], se obtiene el algoritmo de *Retropropagación* o *regla delta generalizada*. Por tanto, la regla delta generalizada se puede resumir de la siguiente manera:

- Para la última capa se aplican las ecuaciones:

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) + \alpha\delta_i^C(n)a_j^{C-1}(n) \text{ para } j = 1, 2,\ldots,n_{c\text{-}1} \text{ y } i = 1, 2,\ldots,n_c \quad (2.22)$$

$$b_i^C(n) = b_i^C(n-1) + \alpha\delta_i^C(n) \text{ para } i = 1, 2, \ldots, n_c \quad (2.23)$$

$$\delta_i^C = (s_i(n) - y_i(n))f'(\sum_{j=1}^{n_c-1} w_{ij}^{C-1}a_j^{C-1} + b_i^C) \quad (2.24)$$

Como es muy frecuente utilizar como función de activación una función sigmoidal, cuya derivada es $f'(x) = f(x)(1 - f(x))$, se utilizará la siguiente ecuación para calcular δ.

$$\delta_i^C = \big(s_i(n) - y_i(n)\big)y_i(n)(1 - y_i(n)) \quad (2.25)$$

- Para el resto de capas se aplican las siguientes ecuaciones:

$$w_{kj}^C(n) = w_{kj}^C(n-1) + \alpha\delta_j^{C+1}(n)a_k^C(n) \quad (2.26)$$

$$b_j^{C+1}(n) = b_j^{C+1}(n-1) + \alpha\delta_j^{C+1}(n) \quad (2.27)$$

$$\delta_j^{C+1} = a_j^{C+1}(1 - a_j^{C+1})\sum_{i=1}^{n_{c+2}}\delta_i^{C+2}w_{ji}^{C+1} \quad (2.28)$$

Teniendo en cuenta esto, el proceso de aprendizaje para el Perceptrón Multicapa está formado por los siguientes pasos:

1. Se asignan los *pesos* y *bias* iniciales. Estos parámetros generalmente se fijan de forma aleatoria.
2. Se presenta un ejemplo de entrenamiento en la entrada de la red y se calcula la salida esperada. Para realizar esto, cuando se recibe el patrón de entrada, las neuronas de la capa de entrada lo propagan hacia la siguiente capa, como se describe en la ecuación (2.17), después las neuronas de la primera capa oculta calculan su nivel de activación utilizando la ecuación (2.18) y lo propagan hacia la siguiente capa. Este proceso se repite hasta que el patrón llega hasta las neuronas de la capa de salida que proporcionan una salida.
3. Se evalúa el error cuadrático medio utilizando la salida esperada y la salida obtenida para ese ejemplo de entrenamiento.
4. Se propaga el error hacia atrás y se van modificando los parámetros de la red. Esto se hace de la siguiente forma:
    4.1. Se calculan los valores δ para todas las neuronas de la capa de salida utilizando la ecuación (2.24).
    4.2. Se van calculando los valores δ del resto de neuronas utilizando la ecuación (2.28), empezando desde la última capa oculta y propagando los valores hacia atrás hasta llegar a la capa de entrada.
    4.3. Se modifican los valores de los pesos y las bias de toda la red.
5. Se repiten los pasos 2, 3 y 4 con todas las instancias del conjunto de entrenamiento y se evalúa el error de entrenamiento cometido.

Los pasos 2, 3, 4 y 5 se repiten durante *n* iteraciones hasta que se cumple una condición de parada. Algunos de los criterios de parada más utilizados son los siguientes:

- El error de entrenamiento permanece estable durante varias iteraciones.
- El error de validación permanece estable durante varias iteraciones.
- El error de validación empieza a aumentar.
- Se establece un número fijo de iteraciones de entrenamiento.

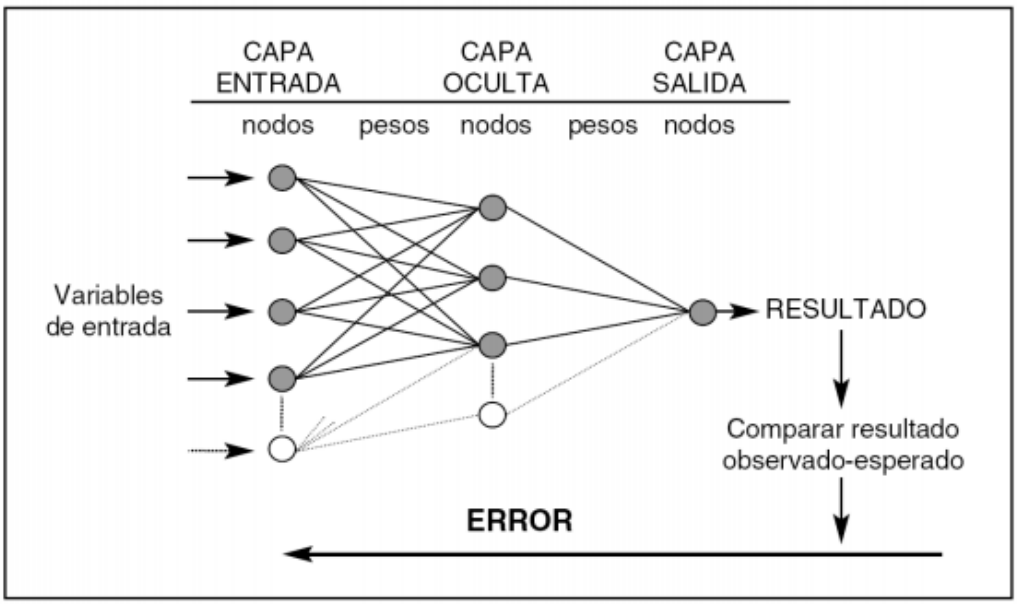

Ilustración 2.18: *Algoritmo de Retropropagación*

Problemas en el algoritmo de aprendizaje

El algoritmo de aprendizaje tiene los siguientes problemas que hay que tener en cuenta a la hora de utilizar el Perceptrón Multicapa:

- *Mínimos locales*: El método del descenso del gradiente sigue la dirección descendiente de la pendiente de la función de error, por lo que habitualmente nos lleva a un mínimo local de dicha función.
- *Parálisis*: Esta situación se presenta porque la función sigmoidal y la tangente hiperbólica, que son las funciones habituales en la función de activación de la neurona, con valores lejanos a cero producen sus valores máximos o mínimos. Cuando se da esta situación, conocida como *saturación*, los parámetros de la red apenas son modificados en el proceso de entrenamiento.

2.4.2.3. Expresividad. Subajuste y sobreajuste

Desde un punto de vista teórico, el aprendizaje de la red de neuronas se debe detener cuando se cumpla $\frac{\partial e(n)}{\partial w} \approx 0$, es decir, cuando no se produce variación en el cálculo de los parámetros de una iteración a otra. Sin embargo, en la práctica el criterio de parada del entrenamiento suele ser realizado estableciendo previamente un número fijo de iteraciones.

Establecer un número fijo de iteraciones para detener el aprendizaje tiene una serie de implicaciones en la capacidad de generalización del modelo final generado, concretamente se pueden dar tres posibles situaciones: *subajuste*, *capacidad de generalización apropiada* o *sobreajuste*.

El subajuste del modelo se da cuando el número de iteraciones con el que ha sido realizado el entrenamiento es bajo para un conjunto de datos concreto y, por tanto, el modelo no se ha podido ajustar a dichos datos lo suficiente. Por ello, el subajuste es un fenómeno indeseable que se debe tratar de evitar.

Por otro lado, el sobreajuste se da cuando el número de iteraciones es demasiado alto y el modelo se ha ajustado de forma excesiva a los datos de entrenamiento. En este caso el error obtenido sobre el conjunto de datos de entrenamiento será muy bajo, pero el modelo producirá malos resultados frente a nuevos patrones. Por este motivo, el sobreajuste también se debe tratar de evitar para generar un modelo que proporcione buenos resultados.

En la *ilustración 2.19* se pueden observar tres modelos distintos generados sobre el mismo conjunto de datos con un número de iteraciones distintas. Como se puede observar, en la imagen de la izquierda el modelo está subajustado ya que no ha sido ajustado lo suficiente, es decir, el error en el conjunto de datos se puede reducir. En la imagen central se observa un modelo generado con un número de iteraciones adecuado, debido a que la función está ajustada de forma correcta sobre el conjunto de datos. Por el contrario, en la imagen de la derecha se puede observar que se produce sobreajuste, el error sobre el conjunto de datos de entrenamiento es nulo y con nuevos patrones el modelo puede obtener un gran error.

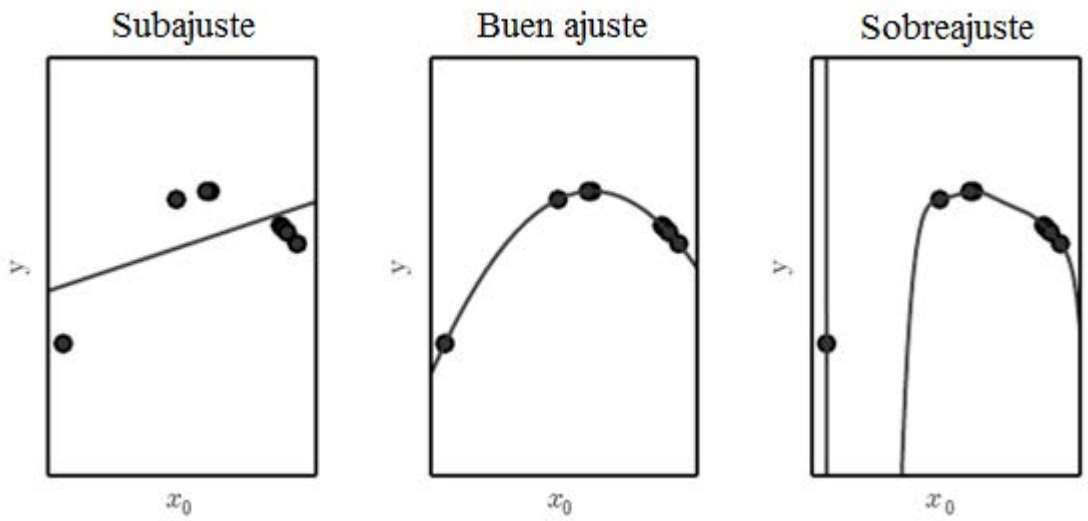

Ilustración 2.19: *Expresividad*

Por tanto, cuando se realiza el proceso de entrenamiento es imprescindible medir la capacidad de generalización, es decir, comprobar que la red pueda responder de forma apropiada ante nuevos patrones que no han sido utilizados anteriormente.

En la *ilustración 2.20* se puede visualizar la evolución habitual de los errores de entrenamiento y generalización en el aprendizaje a medida que pasan las iteraciones. Se puede observar que al principio del entrenamiento, cuando no se han realizado suficientes iteraciones, el error es muy elevado en ambos conjuntos de datos y va disminuyendo a medida que se van actualizando los parámetros de la red con las sucesivas iteraciones. Nótese que el error de generalización permanece estable durante un número de iteraciones concreto, en este punto es donde se debe detener el entrenamiento, ya que si continuamos entrenando la red el error de generalización empezará a aumentar.

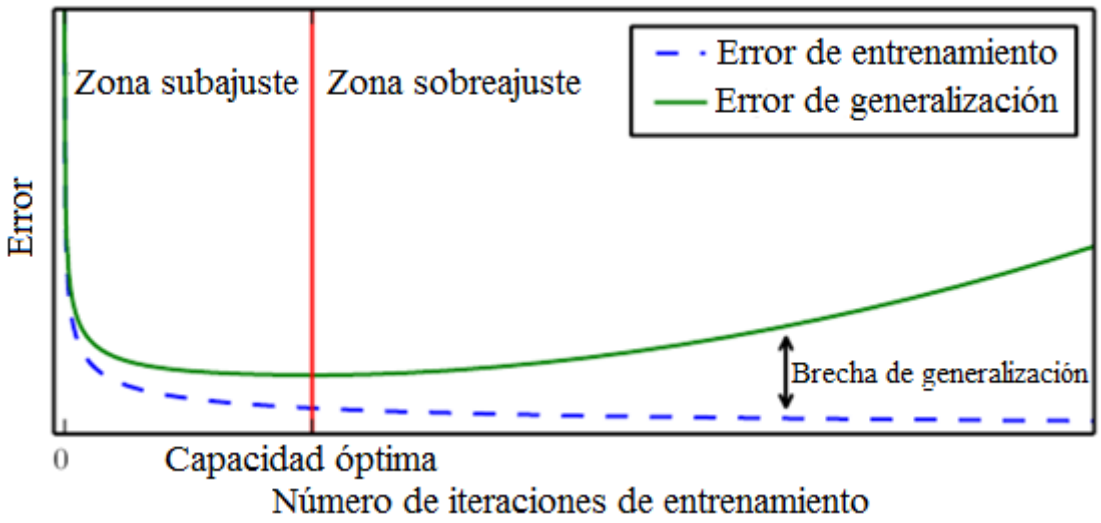

Ilustración 2.20: *Evolución de los errores de entrenamiento y generalización a lo largo del aprendizaje*

2.4.2.4. Regresión y clasificación en Perceptrón Multicapa

El Percentrón Multicapa fue diseñado para resolver la tarea de regresión, esto quiere decir que la salida de la red es siempre un valor continuo. Sin embargo, es posible

convertir todo problema regresión en un problema de clasificación. Para realizar esto, es necesario interpretar la salida de la red.

La forma más sencilla de realizar la conversión es utilizando una única neurona de salida en la red, donde se establece un intervalo de valores para cada clase. Por tanto, una vez que es obtenido el resultado de la red, se comprueba a que intervalo pertenece y entonces es clasificado como la clase asociada a ese intervalo concreto. Así por ejemplo, en un dominio biclase generalmente se utiliza el intervalo [0,0.5] para la primera clase y el intervalo (0.5,1] para la segunda clase.

Otra forma más elaborada de emplear un Perceptrón Multicapa para resolver una tarea de clasificación es poniendo tantas neuronas de salida como clases tengamos en nuestro dominio y cada una estas salidas se asocia a una clase distinta. Para cada dato se escoge la neurona de salida que proporcione un mayor valor y la entrada es clasificada con la clase que tiene asociada esa salida.

2.4.2.5. Autocodificadores

En la actualidad, una de las amplias aplicaciones que tienen las redes de neuronas es la reducción de la dimensionalidad. Para realizar esto, es frecuente utilizar una capa de entrada y una capa de salida con el mismo número de neuronas. Este número corresponde con la cantidad de atributos que tiene el conjunto de atributos. Por tanto, la red de neuronas aprende a producir la misma información que recibe a la entrada. Sin embargo, la clave consiste en utilizar una o varias capas ocultas con un número de neuronas menor que la capa de entrada y salida. Esta arquitectura de la red fuerza la reconstrucción de los datos de entrada en las neuronas de la capa oculta, de manera que aprenden a capturar la máxima cantidad de información posible y sus activaciones pueden ser utilizadas como nuevos atributos con un menor número de dimensiones. Una red de neuronas con esta arquitectura se le conoce con el nombre de *autocodificador*.

Como se puede observar en la *ilustración 2.21*, una vez entrenada la red, es posible dividirla en dos partes: una primera que utiliza la capa oculta como salida y que se encarga de realizar una compresión de los datos, y una segunda que utiliza la capa oculta como entrada que se encarga de la descompresión de los datos.

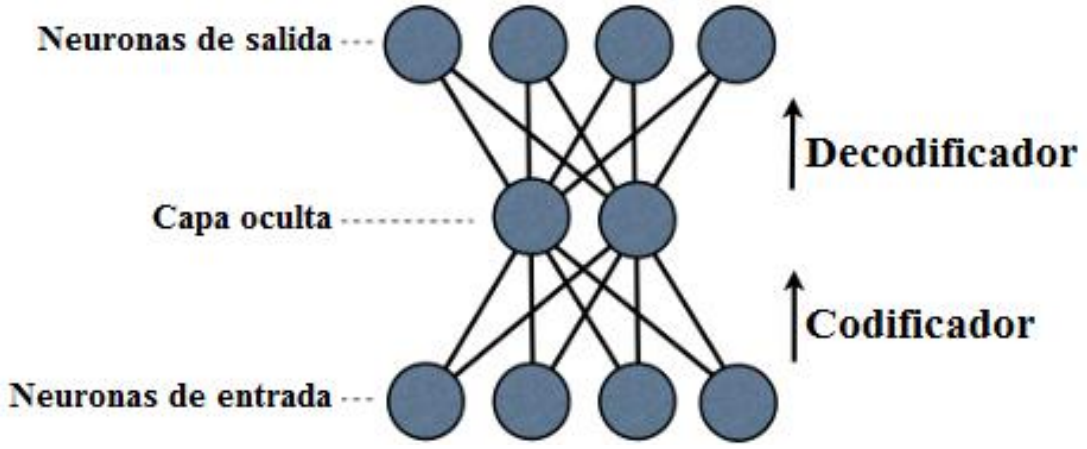

Ilustración 2.21: *Autocodificador*

2.5. Comparación de las soluciones presentadas

En este apartado se muestra una comparación de los métodos que son empleados a lo largo del presente trabajo. La primera técnica para hacer la reducción de la dimensionalidad es una red de neuronas artificial y la segunda técnica es PCA.

La principal diferencia entre ambas versiones es que PCA realiza una transformación lineal de los datos, mientras que una red de neuronas es capaz de encontrar relaciones más complejas en los datos, es decir, produce una transformación no lineal. Aunque, como se hace en este trabajo, también es posible realizar una transformación lineal de los datos con una red de neuronas.

PCA siempre es una técnica no supervisada. Por otro lado, en una red de neuronas se puede utilizar para la reducción de dimensionalidad con una arquitectura supervisada o una arquitectura no supervisada: un autocodificador es no supervisado, pero el esquema implementado en este trabajo, con una única neurona de salida, es supervisado.

Otra diferencia fundamental entre ambas aproximaciones es que PCA calcula sus componentes principales en función de la varianza de los datos, mientras que los pesos de una red de neuronas son ajustados en función del error en la salida de la red, es decir, su objetivo es minimizar el error de reconstrucción.

Por último, de forma general PCA necesita menos ejemplos de entrenamiento para producir un mejor resultado. En el caso de la red de neuronas se suelen necesitar más ejemplos, ya que de lo contrario se puede producir sobreaprendizaje al existir muchos grados de libertad.

PCA	Redes de neuronas
Lineal	No lineal Lineal
No supervisado	No supervisado Supervisado
Pocos ejemplos de entrenamiento	Más ejemplos de entrenamiento
Componentes calculados en función de la varianza de los datos	Pesos ajustados en función del error que se produce en la salida de la red

Tabla 2.2: *Resumen de la comparación de PCA y redes de neuronas artificiales*

2.6. K-nearest neighbors (KNN)

El método *K-nearest neighbors* (*KNN*) es un método de aprendizaje basado en instancias. Por tanto, en la fase de aprendizaje lo único que hace es almacenar todos los datos de entrenamiento.

Una vez almacenados los datos de entrenamiento, cuando llega un nuevo dato cuyo valor es desconocido, este es comparado con todos los demás que se tienen registrados y se genera una respuesta teniendo en cuenta los k datos más parecidos. Frecuentemente, la función de similitud más utilizada es la inversa de la distancia euclídea. La distancia euclídea se calcula utilizando la ecuación (2.29).

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2} \quad (2.29)$$

Donde:

- x_i es el dato a clasificar.
- x_j es el dato almacenado.
- x_{ik} es el valor del atributo *k* para el ejemplo x_i.
- x_{jk} es el valor del atributo *k* para el ejemplo x_j.

Así, la similitud es calculada con la siguiente ecuación:

$$s(x_i, x_j) = \frac{1}{1+d(x_i,x_j)} \quad (2.30)$$

La tarea de clasificación consiste en calcular la similitud entre el ejemplo a clasificar x_i y todos los ejemplos de entrenamiento x_j que se tienen almacenados. A continuación, se seleccionan los *k* vecinos más cercanos, es decir, aquellos que tienen un mayor valor en la función s. Por último, se calcula la clase de salida como la clase mayoritaria de los *k* vecinos más cercanos.

Es importante destacar que para realizar la clasificación se debe definir previamente el parámetro *k*. Para realizar esto, no existe ningún método que indique el valor exacto de *k* que debe ser utilizado, por lo que suele ser obtenido de forma experimental.

Capítulo 3

Marco regulador

3.1. Marco regulador

En este apartado se realiza un estudio detallado del marco regulador que afecta al presente trabajo. El estudio será realizado en los siguientes ámbitos:

- Lenguaje de programación R.
- Entorno de programación RStudio.
- Herramientas de Microsoft.
- Conjunto de datos.

3.1.1. Lenguaje de programación R

R es un lenguaje de programación de código abierto y es un paquete de GNU. Se distribuye de forma gratuita bajo la licencia *GNU General Public License* (*GNU GPL*).

La licencia GNU GPL es ampliamente utilizada en el software de código abierto y garantiza que los usuarios finales tengan la libertad para ejecutar, estudiar, compartir y modificar el software de forma libre y gratuita. La licencia protege los derechos de los desarrolladores de dos formas:

- Reconoce el copyright del desarrollador del programa.
- Le ofrecen a los usuarios finales el permiso legal para copiar, distribuir y/o modificar el software. Sin embargo, GNU GPL es una licencia *copyleft*, que significa que el trabajo derivado solamente puede ser distribuido utilizando los mismos términos de licencia.

Además, esta licencia no asegura la garantía del software, pero para proteger a los desarrolladores si el software es modificado y redistribuido por alguien, se asegura que los receptores hayan sido notificados de que no poseen la copia original.

3.1.2. Entorno de programación RStudio

El entorno de programación llamado RStudio está disponible bajo la licencia *GNU Affero General Public License (Affero GPL o AGPL)*. La licencia AGPL es prácticamente idéntica a la licencia GPU GNL, la única diferencia entre ellas es que AGPL integra una nueva cláusula que obliga a proporcionar el código fuente del software si éste ofrece servicios en una red de ordenadores. Por tanto, para RStudio se ofrecen dos versiones de forma gratuita:

- RStudio Desktop: Tiene integradas todas las herramientas para R.

- RStudio Server: En esta versión también están integradas todas las herramientas para R y, además, ofrece la posibilidad de acceder desde un navegador web.

Adicionalmente, cabe destacar que en RStudio también podemos encontrar dos versiones comerciales que incluyen muchas más características que las anteriormente citadas.

3.1.3. Herramientas de Microsoft

Los productos de Microsoft no son de código abierto, ni gratuitos y ofrece tres tipos de licencias. A continuación, se realiza una descripción de este tipo de licencias:

- Licencia *OEM*: Este tipo de licencia es la más habitual en los equipos cliente, ya que son licencias que vienen incluidas en los dispositivos cuando son comprados de forma legal. Generalmente, esta licencia suele encontrarse adherida en el equipo en forma de pegatina con hologramas y tiene la característica que no puede ser utilizada en otro equipo que no sea el original.
- Licencia *Retail*: Esta licencia es obtenida cuando la compra es realizada en una tienda o por la web oficial. En este caso, la licencia viene en DVD o en una imagen con formato ISO para realizar la posterior instalación. Dentro de esta licencia existen dos opciones, por un lado, podemos adquirir una licencia completa que permite la instalación del software en cualquier equipo y, por otro lado, existe una licencia de actualización que permite hacer actualizaciones del software a la versión más reciente.
- Licencia por *volumen*: Esta licencia está enfocada para llevar a cabo la instalación del software en gran cantidad de equipos, por tanto, es muy recomendable para grandes empresas que quieran realizar la instalación en más de 100 usuarios.

3.1.4. Conjunto de datos

En este estudio al trabajar con datos es muy importante cumplir con la *Ley Orgánica de Protección de Datos (LOPD)*. La Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal es una ley orgánica española que tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor, intimidad y privacidad personal y familiar.

El dominio conocido con el nombre de *doughnut* (la descripción detallada se realiza en el apartado *6.2. Dominio Doughnut*) ha sido generado por los tutores y el desarrollador del presente proyecto. El resto de los conjuntos de datos utilizados han sido obtenidos en el repositorio de datos llamado *UCI*. Este repositorio contiene multitud de bases de datos y dominios que han sido donados para realizar tareas de investigación en algoritmos de aprendizaje automático y pueden ser usados de forma libre, dicho repositorio está alojado en la siguiente página web:
http://archive.ics.uci.edu/ml/index.html

3.1.5. Conclusiones del análisis del marco regulador

Una vez realizado el análisis del marco regulador, se va a detallar las licencias utilizadas en el presente trabajo para mostrar que se ha respetado la legislación vigente en todos los casos.

En el caso de R se puede utilizar sin restricciones al ser gratuito y de código abierto. La versión utilizada de RStudio será la versión gratuita RStuido Desktop, ya que contiene todas las funcionalidades de R y para el trabajo desarrollado en este caso es suficiente. Con Windows 10 Home, Microsoft Word 2010 y Microsoft Excel 2010 será utilizada una licencia OEM previamente instalada en el equipo. Los diagramas realizados con Microsoft Project han sido bajo la licencia de la universidad, ya que se han utilizado los equipos de la biblioteca.

Por otro lado, se ha respetado la Ley Orgánica de Protección de Datos al haber utilizado datos que han sido donados y pueden ser usados de forma libre.

También es importante destacar que se ha tenido en cuenta la legislación sobre los derechos de autor que en España se establece dentro de la Ley 22/1987 y que, posteriormente, ha recibido algunas modificaciones.

3.2. Estándares técnicos

El único estándar utilizado en el desarrollo del trabajo ha sido en la obtención de requisitos. En este caso se ha utilizado el estándar definido por Métrica versión 3, que puede ser utilizada de forma libre y es propiedad intelectual del Ministerio de Hacienda y Administraciones Públicas.

3.3. Rangos salariales

Este trabajo se asume que está sujeto al *Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos*. Este convenio, publicado en el Boletín Oficial del Estado (BOE), es de aplicación en todo el territorio español y su vigencia se extiende hasta el 31 de diciembre de 2017. En la siguiente tabla se muestran los niveles salariales de dicho convenio.

	Mes × 14	Anual
Nivel 1. Licenciados y titulados 2.º y 3.er ciclo universitario y Analista	1.687,02	23.618,28
Nivel 2. Diplomados y titulados 1.er ciclo universitario. Jefe Superior	1.253,16	17.544,24
Nivel 3. Técnico de cálculo o diseño, Jefe de 1.ª y Programador de ordenador	1.208,40	16.917,60
Nivel 4. Delineante-Proyectista, Jefe de 2.ª y Programador de maq. Auxiliares	1.107,87	15.510,18
Nivel 5. Delineante, Técnico de 1.ª, Oficial 1.ª Admtvo. y Operador de ordenador	968,23	13.555,22
Nivel 6. Dibujante, Técnico de 2.ª, Oficial 2.ª Admtvo., Perforista, Grabador y Conserje	834,17	11.678,38
Nivel 7. Telefonista-Recepcionista, Oficial 1.ª oficios varios, y Vigilante	806,20	11.286,80
Nivel 8. Auxiliar Técnico, Auxiliar Admtvo., Telefonista, Ordenanza, Personal de limpieza y Oficial 2.ª oficios varios	750,38	10.505,32
Nivel 9. Ayudante oficios varios	698,24	9.775,36

Ilustración 3.1: *Rangos salariales del convenio*

En el convenio se establece que la jornada ordinaria máxima de trabajo efectivo, en cómputo anual, es de 1.800 horas. Teniendo en cuenta el número de horas anterior y las funciones que puede desarrollar cada cargo, se obtienen los precios mínimos que debe cobrar cada trabajador. Sin embargo, estos precios se han decidido incrementar en, aproximadamente, 10 euros al considerarse bajos. Además de esto, se ha incrementado

al alza el salario de cada puesto para poder cubrir los gastos asociados a los impuestos (seguridad social, etc.). Por tanto, en la *Tabla 3.1* se muestra la siguiente información:

- *Cargo*: Indica la función que se desempeña en el proyecto.
- *Nivel*: Indica el nivel asignado del convenio. Para hacer esta asignación se ha tenido en cuenta la descripción que se hace de cada puesto en el convenio.
- *Salario por hora (convenio)*: Es el salario mínimo por hora que debe cobrar cada cargo. Este valor ha sido calculado con los precios indicados en el convenio y teniendo en cuenta que en un año existen 1.800 horas laborales.
- *Salario por hora (proyecto)*: Es el salario por hora que se le ha asignado a cada cargo para el presente proyecto. Se ha tomado la decisión de incrementar el salario establecido en el convenio en 10 euros/hora, aproximadamente.
- *Coste por hora (proyecto)*: Es el coste total por hora que supone cada cargo. En este precio están incluidos los impuestos y el salario. Será el precio final utilizado para calcular el presupuesto.

Cargo	Nivel	Salario por hora (convenio)	Salario por hora (proyecto)	Coste por hora (proyecto)
Jefe de proyecto	Nivel 1	13,12 €/hora	23 €/hora	25 €/hora
Analista	Nivel 1	13,12 €/hora	23 €/hora	25 €/hora
Diseñador	Nivel 3	9,40 €/hora	19 €/hora	20 €/hora
Gestor de calidad y pruebas	Nivel 3	9,40 €/hora	19 €/hora	20 €/hora
Documentación	Nivel 3	9,40 €/hora	19 €/hora	20 €/hora
Programador	Nivel 3	9,40 €/hora	19 €/hora	20 €/hora

Tabla 3.1: *Resumen de los rangos salariales del proyecto*

Capítulo 4

Entorno socio-económico

4.1. Planificación

En la siguiente tabla se muestra la planificación inicial estimada para la realización del proyecto. Como se puede observar, se estima la duración del proyecto en 9 meses (desde Septiembre hasta Mayo, ambos incluidos) para poder realizar la entrega del mismo en la convocatoria de Junio. Por ello, se establece la entrega final para el día 15 de Junio de 2017. La planificación inicial del proyecto también se muestra de forma gráfica en la *ilustración 4.1*.

Nombre de tarea	Duración	Comienzo	Fin
Establecimiento de objetivos	**11 días**	**Lun 05/09/16**	**Sáb 17/09/16**
Entorno socio-económico	**5 días**	**Lun 19/09/16**	**Vie 23/09/16**
Planificación	2 días	Lun 19/09/16	Mar 20/09/16
Presupuesto	2 días	Mié 21/09/16	Jue 22/09/16
Análisis impacto socio-económico	1 día	Vie 23/09/16	Vie 23/09/16
Marco regulador	**6 días**	**Lun 26/09/16**	**Sáb 01/10/16**
Análisis estado del arte	**86 días**	**Lun 03/10/16**	**Sáb 28/01/17**
Reducción de dimensionalidad	36 días	Lun 03/10/16	Lun 21/11/16
PCA	20 días	Mar 22/11/16	Lun 19/12/16
Aprendizaje automático	17 días	Mar 20/12/16	Mié 11/01/17
Redes de neuronas artificiales	9 días	Jue 12/01/17	Mar 24/01/17
KNN	3 días	Mié 25/01/17	Vie 27/01/17
Diseño y alternativas	**11 días**	**Lun 30/01/17**	**Sáb 11/02/17**
Implementación y pruebas	**16 días**	**Lun 13/02/17**	**Sáb 04/03/17**
Realización experimentos	**26 días**	**Lun 06/03/17**	**Sáb 08/04/17**
Dominio Doughnut	6 días	Lun 06/03/17	Sáb 11/03/17
Dominio Spambase	6 días	Lun 13/03/17	Sáb 18/03/17
Dominio German	6 días	Lun 20/0317	Lun 27/03/17
Dominio Splice	10 días	Mar 28/03/17	Sáb 08/04/17
Análisis de resultados	**16 días**	**Lun 10/04/17**	**Sáb 29/04/17**
Dominio Doughnut	4 días	Lun 10/04/17	Jue 13/04/17
Dominio Spambase	4 días	Vie 14/04/17	Mié 19/04/17
Dominio German	4 días	Jue 20/04/17	Mar 25/04/17
Dominio Splice	4 días	Mié 26/04/17	Sáb 29/04/17
Conclusiones	**6 días**	**Lun 01/05/17**	**Sáb 06/05/17**
Anexo I: Abstract	**6 días**	**Lun 08/05/17**	**Lun 15/05/17**
Entrega	**0 días**	**Jue 15/06/17**	**Jue 15/06/17**

Tabla 4.1: *Planificación inicial del poyecto*

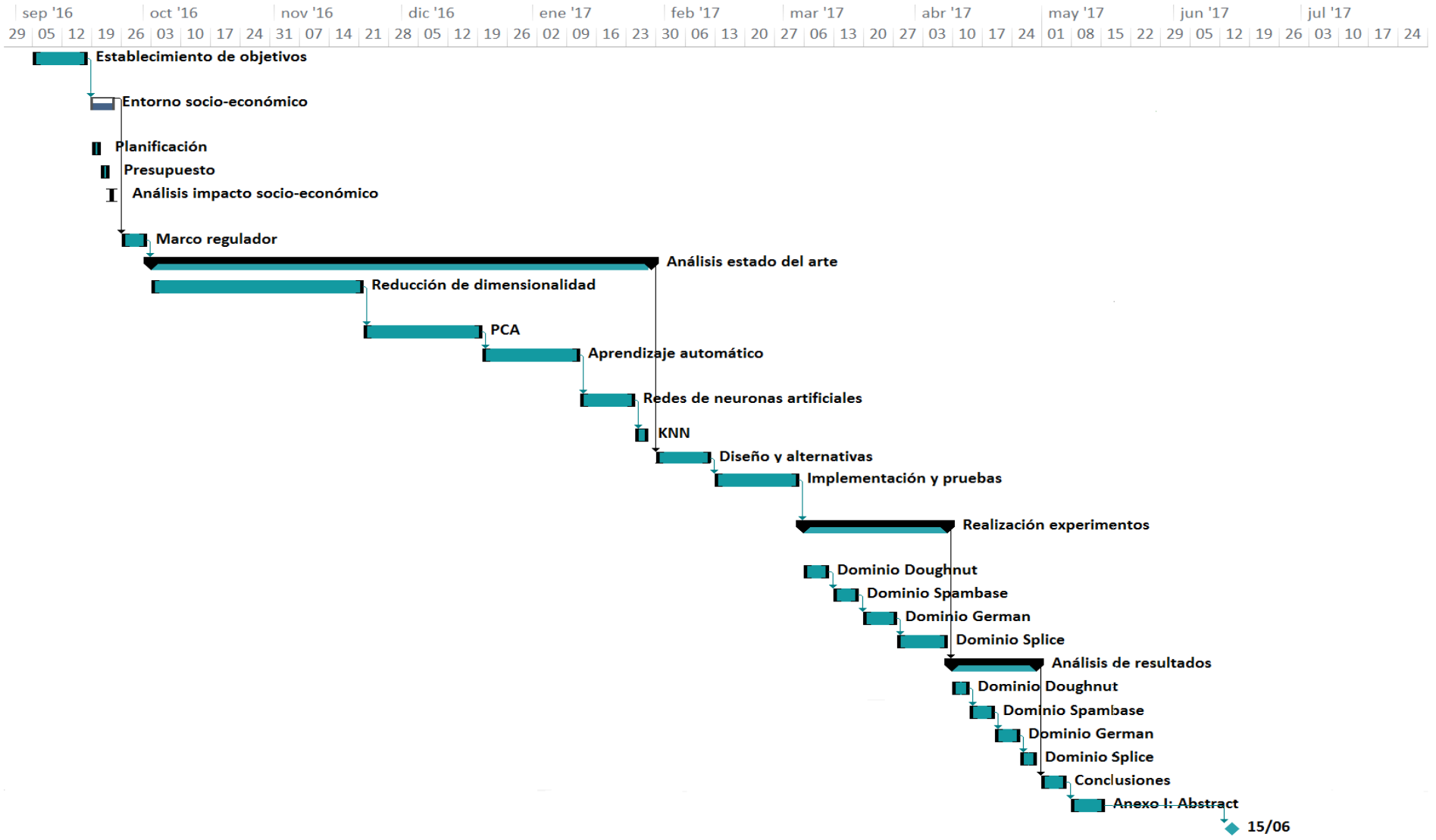

Ilustración 4.1: *Planificación inicial del proyecto*

4.2. Presupuesto

En este apartado se detalla el presupuesto de realización del proyecto, para llevarlo a cabo se ha tenido en cuenta la planificación realizada en el apartado anterior y los costes de personal obtenidos en el capítulo anterior.

4.2.1. Costes de personal

El desarrollo del trabajo ha sido realizado por una única persona, pero se han desempeñado diversas funciones a lo largo del trabajo realizado. A continuación, se muestra cada función desempeñada con su correspondiente precio asociado.

Cargo	Persona a cargo	Coste por hora
Jefe de proyecto	José María Valls Ferrán Ricardo Aler Mur Álvaro Soriano Maganto	25 €/hora
Analista	Álvaro Soriano Maganto	25 €/hora
Diseñador	Álvaro Soriano Maganto	20 €/hora
Gestor de calidad y pruebas	Álvaro Soriano Maganto	20 €/hora
Documentación	Álvaro Soriano Maganto	20 €/hora
Programador	Álvaro Soriano Maganto	20 €/hora

Tabla 4.2: *Resumen de personal del proyecto*

En la *Tabla 4.4* se muestra el número de horas estimado a realizar en cada función desempeñada en el proyecto. En la *Tabla 4.3* se han calculado los gastos por función. Para realizar esto, se ha utilizado el número de horas estimado y el coste por hora. En esta tabla se observa también que el gasto total en concepto de personal asciende a la cantidad de 16.460 €.

Cargo	Total horas	Coste por hora	Coste total
Jefe de proyecto	38	25 €/hora	950 €
Analista	238	25 €/hora	5.950 €
Diseñador	30	20 €/hora	600 €
Gestor de calidad y pruebas	35	20 €/hora	700 €
Documentación	338	20 €/hora	6.760 €
Programador	75	20 €/hora	1.500 €
TOTAL			**16.460 €**

Tabla 4.3: *Coste total de personal*

Actividad	Jefe de proyecto	Analista	Diseñador	Gestor de calidad y pruebas	Documentación	Programador
Establecimiento de objetivos	20	20	0	1	1	0
Análisis del estado del arte	1	148	0	1	194	0
Marco regulador	0	20	0	2	2	0
Entorno socio-económico	11	2	2	1	3	1
Diseño y análisis de alternativas	4	11	26	3	0	0
Implementación y pruebas	0	1	2	24	2	35
Realización experimentos	0	0	0	1	64	39
Análisis de resultados	2	32	0	1	29	0
Conclusiones	0	3	0	1	20	0
Anexo	0	1	0	0	23	0
Horas totales	**38**	**238**	**30**	**35**	**338**	**75**

Tabla 4.4: *Reparto de horas del proyecto*

Las horas estimadas de la *Tabla 4.4* han sido obtenidas utilizando la planificación inicial y teniendo en cuenta que se trabajarán 4 horas diarias, aproximadamente.

4.2.2. Equipos informáticos

Para estimar el costo de los equipos informáticos se ha calculado la amortización en el periodo de tiempo que han sido utilizados. Es importante destacar que para realizar el cálculo se ha supuesto una vida útil de 2 años (24 meses) en los equipos informáticos.

Descripción	Precio unitario	Unidades	Coste/mes	Número de meses	Precio total
Ordenador	1.200 €	1	50 €/mes	9	450 €
TOTAL					**450 €**

Tabla 4.5: *Gasto en equipos informáticos*

4.2.3. Herramientas de software

En la siguiente tabla se muestran los gastos de software externo utilizado durante el desarrollo del proyecto.

Descripción	Unidades	Coste/unidad	Precio total
Microsoft Windows 10 Home	1	0 €	0 €
Microsoft Word 2010	1	0 €	0 €
Microsoft Excel 2010	1	0 €	0 €
Microsoft Project	1	0 €	0 €
RStudio Desktop	1	0 €	0 €
TOTAL			**0 €**

Tabla 4.6: *Gasto en herramientas de software*

El gasto total en herramientas de software es de 0 €. Esto es debido a que para Microsoft Windows 10 Home, Microsoft Word 2010 y Microsoft Excel 2010 se obtuvo una licencia OEM con la adquisición del ordenador. Microsoft Project ha sido utilizado con la licencia de la universidad y la licencia de RStudio Desktop es gratuita.

4.2.4. Material fungible

En el coste de material fungible se incluyen los gastos de oficina (bolígrafos, papel, impresiones, etc.) y una memoria USB utilizada para transportar la información. A continuación, se muestran estos gastos.

Descripción	Unidades	Coste/unidad	Precio total
Material de oficina	1	30 €	30 €
Memoria USB	1	7,90 €	7,90 €
TOTAL			**37,9 €**

Tabla 4.7: *Gasto en material fungible*

4.2.5. Viajes y dietas

En la siguiente tabla se observan los gastos asociados con el transporte y las comidas. Aunque gran parte del trabajo será realizado a distancia, será necesario organizar reuniones periódicas entre los jefes de proyecto.

Descripción	Precio total
Gasolina	300 €
Comidas	50 €
TOTAL	**350 €**

Tabla 4.8: *Gasto en viajes y comidas*

4.2.6. Costes indirectos

En esta categoría se incluyen los gastos relacionados con el lugar de trabajo durante el tiempo de desarrollo del proyecto.

Descripción	Precio total
Electricidad	200 €
Agua	50 €
Conexión a internet	150 €
TOTAL	**400 €**

Tabla 4.9: *Gastos de costes indirectos*

4.2.7. Resumen de costes totales

En la *Tabla 4.10* se muestran un resumen con todos los gastos totales expuestos en los anteriores apartados. Se puede observar que el gasto total del proyecto asciende a la cantidad de 17.697,9 €.

Concepto	Precio total
Gasto de personal	16.460 €
Equipos informáticos	450 €
Herramientas de software	0 €
Material fungible	37,9 €
Viajes y dietas	350 €
Costes indirectos	400 €
TOTAL	**17.697,9 €**

Tabla 4.10: *Resumen de costes totales*

4.2.8. Coste total del proyecto

Teniendo en cuenta que el gasto total del proyecto es de 17.697,9 €, se va a proceder a realizar el cálculo del presupuesto final.

Concepto	Cantidad
Gasto total	17.697,9 €
Riesgo (15% del gasto total)	2.654,69 €
Beneficio (20% del gasto total)	3.539,58 €
TOTAL SIN I.V.A.	**23.892, 17 €**
I.V.A. (21%)	5.017,36 €
TOTAL CON I.V.A.	**28.909,53 €**

Tabla 4.11: *Coste total del proyecto*

El presupuesto total de este proyecto asciende a la cantidad total de:

28.909,53 €

Veintiocho mil novecientos nueve euros con cincuenta y tres céntimos

Fdo. D. Álvaro Soriano Maganto

El ingeniero proyectista

4.3. Análisis del impacto socio-económico

Como ya se comentó anteriormente, algunas de las aplicaciones que tienen los algoritmos de reducción de dimensionalidad son los siguientes:

- Categorización de textos.
- Recuperación de imágenes.
- Reconocimiento de caras y dígitos.
- Detección de intrusiones informáticas.
- Clasificación de enfermedades.

Además de estas utilidades, cabe destacar que en la actualidad las empresas y organizaciones son capaces de reunir gran cantidad de información y tienen la necesidad de llevar a cabo un análisis para extraer conocimiento con diversos fines. Algunas empresas utilizan esta información para prestar servicios, así por ejemplo Google Maps utiliza la información para encontrar la ruta óptima o Google Search para mejorar la calidad de las búsquedas.

También es muy común analizar la información en el mundo empresarial, en áreas como puede ser el marketing o la banca, donde a menudo este conocimiento es utilizado para obtener ventaja frente a sus competidores.

Generalmente, el análisis de la información suele ser realizado mediante técnicas de aprendizaje automático, ya que hoy en día se genera gran cantidad de información y sería imposible que tal cantidad de datos fuese analizada por personas en tiempo real.

Actualmente son generados aproximadamente 2,5 exabytes de información. Para almacenar esta cantidad de información se necesitarían 10 millones de discos blu-ray. Adicionalmente, la cantidad de datos almacenada por segundo cada vez es más elevada, por ejemplo el 90% de los datos mundiales actuales han sido producidos en los últimos dos años. Además del volumen de información, suele existir una gran variedad de datos y requieren que la velocidad de respuesta sea lo demasiado rápida como para lograr obtener conclusiones en el momento apropiado. Un problema con estas características es conocido con el nombre de *Big Data*.

Por ello, los algoritmos de aprendizaje automático han cobrado una gran importancia y tienen un gran impacto social y económico. El algoritmo de reducción de la dimensionalidad estudiado está pensado para mejorar el rendimiento de los algoritmos de aprendizaje automático y, por tanto, puede tener gran impacto tanto social como económico.

Capítulo 5

Diseño de la solución

5.1. Descripción general de la solución

Los métodos elegidos para realizar el estudio de reducción de la dimensionalidad son los basados en extracción de características, concretamente se va a realizar un estudio de la viabilidad de un esquema supervisado basado en redes de neuronas artificiales utilizando el Perceptrón Multicapa. Para evaluar la calidad de la transformación de los datos será utilizado KNN. Adicionalmente, será realizada una comparación entre el esquema propuesto, basado en redes de neuronas artificiales, con PCA, ya que es un algoritmo ampliamente utilizado y conocido.

En este apartado se va a explicar en detalle el procedimiento elegido para hacer el análisis del nuevo método de reducción de la dimensionalidad, así como la comparación con PCA.

En primer lugar, cabe destacar que los datos originales de los dominios empleados en el estudio han sido sometidos a un preprocesado de datos, antes de utilizar las técnicas de reducción de dimensionalidad propiamente dichas. Las técnicas empleadas en este trabajo han sido la numerización, normalización y aleatorización de los datos.

El primer paso de preprocesado realizado consiste en la numerización de los datos, realizar esto es imprescindible ya que tanto PCA como Perceptrón Multicapa son técnicas que no son capaces de trabajar con atributos nominales. Una vez que todos los atributos son de tipo numérico, se ha realizado una normalización de los datos para evitar el efecto de parálisis en las neuronas. Por último, se realiza una aleatorización de los datos separándolos en dos conjuntos de datos, utilizando un 60% de los datos para el conjunto de entrenamiento y el resto de datos como conjunto de test.

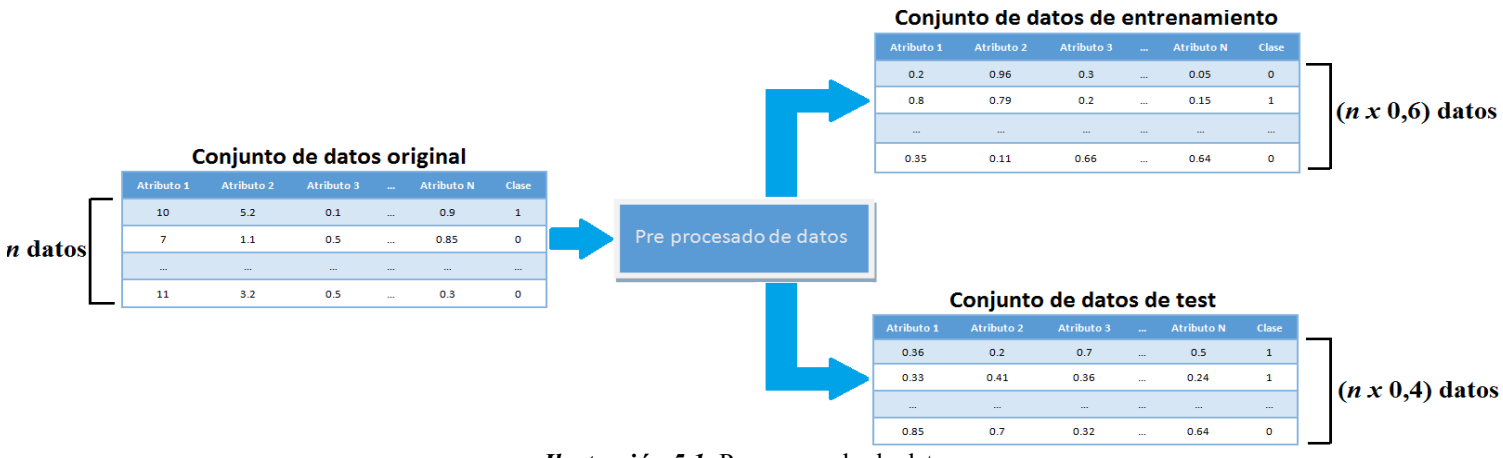

Ilustración 5.1: Preprocesado de datos

Es importante mencionar que en este caso no se ha utilizado un conjunto de datos para validación y que la técnica de evaluación de los resultados que ha sido empleada es *validación simple*.

Con los datos preparados es posible comenzar a aplicar el procedimiento para la reducción de la dimensionalidad. El proceso para llevar a cabo la experimentación puede ser dividido en tres bloques:

- Fase 1: Reducción de la dimensionalidad aplicando el Perceptrón Multicapa y evaluación mediante KNN.
- Fase 2: Evaluación de la precisión de KNN sobre los datos originales.
- Fase 3: Reducción de la dimensionalidad aplicando PCA y evaluación mediante KNN.

Hay que tener en cuenta que para cada dominio se realizarán diferentes experimentos, reduciendo el número de dimensiones en diferentes tamaños para encontrar el valor óptimo de dimensiones en cada caso. Además, como se realizará un estudio comparativo con PCA, en cada dominio se llevarán a cabo experimentos con el mismo número de componentes (en el caso de PCA) y neuronas en la capa oculta (en caso del esquema utilizando el Perceptrón Multicapa).

Además, el parámetro *k* que se debe proporcionar al algoritmo KNN será obtenido de forma experimental, es decir, mediante prueba y error. Por este motivo, se realizarán diferentes pruebas variando dicho parámetro, los valores de *k* que se utilizarán en todos los dominios serán 1, 3, 5 y 7.

5.1.1. Fase 1: Reducción de la dimensionalidad aplicando el Perceptrón Multicapa y evaluación mediante KNN

5.1.1.1. Diseño de la red

Como ya se comentó anteriormente, el primer paso para utilizar un modelo basado en el Perceptrón Multicapa consiste en definir su arquitectura. En este caso, las elecciones de diseño tomadas son las siguientes:

- El número de neuronas de la capa de entrada será igual al número de atributos que tiene el conjunto de datos, es decir, a cada atributo le corresponde una neurona de entrada.
- Habrá una única neurona de salida. En este caso los problemas a resolver serán de clasificación biclase, por lo que la salida del Perceptrón Multicapa debe ser interpretada de la siguiente forma: si el resultado de la red está en el intervalo [0, 0.5], se considerará que el dato pertenece a la primera clase; mientras que si el resultado está en el intervalo (0.5, 1], se considerará que el dato pertenece a la segunda clase.
- Habrá una única capa oculta. El número de neuronas de la capa oculta corresponderá con el número de dimensiones del nuevo conjunto de datos transformado, por lo que en ningún caso será mayor que el número de neuronas de la capa de entrada. En cada dominio se realizarán experimentos con distinto número de neuronas en la capa oculta para encontrar el valor óptimo.
- La función de activación elegida para todas las neuronas será la función sigmoidal.

5.1.1.2. Entrenamiento de la red

El entrenamiento de la red de neuronas será realizado utilizando el algoritmo de *Retropropagación* (este algoritmo es explicado en el apartado *2.4.2.2. Entrenamiento de Perceptrón Multicapa*).

En este caso, el criterio de parada utilizado para detener el aprendizaje consiste en establecer previamente un número fijo de iteraciones, por lo que para evitar el subajuste y sobreajuste del modelo (véase *2.4.2.3. Expresividad. Subajuste y sobreajuste*) se realizarán dos entrenamientos de la red independientes, utilizando números distintos de iteraciones en cada entrenamiento. Por tanto, se realizarán los mismos experimentos con 100 y con 1.000 iteraciones de entrenamiento.

Como los pesos del Perceptrón Multicapa son iniciados aleatoriamente es posible que la calidad del modelo obtenida varíe dependiendo de dicha inicialización, por este motivo la tasa de acierto se calculará como la media del resultado de la repetición del mismo experimento, concretamente será la media de cinco repeticiones. También, hay que tener presente la existencia de los mínimos locales en la función de error que se intenta minimizar, para intentar evitar este fenómeno indeseable todos los experimentos serán repetidos 50 veces y serán agrupados en grupos de 10 para escoger las 5 mejores repeticiones. Por tanto, la tasa media será realmente la media de las cinco mejores repeticiones. Este proceso que puede resultar costoso de comprender se explica en detalle más adelante.

5.1.1.3. Evaluación de la red de neuronas

Para elegir las mejores repeticiones será utilizada la tasa de acierto medida en la salida de la red sobre el conjunto de datos de entrenamiento. Por este motivo, una vez entrenada la red de neuronas, el siguiente paso consistirá en medir la tasa de acierto sobre el conjunto de datos de entrenamiento en la salida de la red. Cabe destacar que esta tasa de acierto no será utilizada en el estudio, solo a la hora de elegir los mejores experimentos, por lo que no se mostrarán estos resultados a lo largo del presente documento.

5.1.1.4. Evaluación de las activaciones de las neuronas de la capa oculta

Como los atributos del nuevo conjunto de datos corresponden con las activaciones de las neuronas de la capa oculta, se debe aplicar KNN a las activaciones de dichas neuronas para llevar a cabo la evaluación de la reducción de la dimensionalidad. En este caso, se ha decidido realizar tres evaluaciones diferentes y son las siguientes:

1. Evaluación de la transformación lineal realizada antes de aplicar la función sigmoidal de la capa oculta.
2. Evaluación de la transformación lineal normalizada.
3. Evaluación de la transformación realizada por la capa oculta, cuando se ha aplicado la función sigmoidal.

A continuación, se va a realizar una descripción detallada del procedimiento necesario para realizar las tres evaluaciones.

Evaluación de la transformación lineal realizada antes de aplicar la función sigmoidal de la capa oculta

Una vez realizado el entrenamiento de la red, se obtendrán las activaciones de las neuronas de la capa oculta antes de aplicar la función de activación sigmoidal, tal y como se detalla en la ecuación (2.16) del apartado *2.4.1.1. La neurona artificial*. En otras palabras, se calculará la transformación lineal que realiza la capa oculta de la red. Para obtener dichas activaciones se debe resolver la siguiente ecuación:

$$Y = XW^T \tag{5.1}$$

Donde:

- Y es la matriz que contiene los resultados de la transformación lineal realizada por las neuronas de la capa oculta, es decir, contiene los datos con un número reducido de dimensiones.
- W es la matriz que contiene las bias de cada neurona de la capa oculta y los pesos de las conexiones entre las neuronas de la capa de entrada y las neuronas de la capa oculta.
- X es la matriz que contiene los datos originales y una columna adicional con el valor 1, esta columna se añade para poder aplicar las bias de las neuronas.

Es importante destacar que se calculan por separado las activaciones que producen los datos de entrenamiento y las activaciones que producen los datos de test. Por tanto, la ecuación (5.1) debe ser aplicada dos veces, por un lado, la matriz X corresponderá con la matriz de datos de entrenamiento X_e generando una matriz que denominaremos Y_e y, por otro lado, la matriz X corresponderá con la matriz de datos de test X_t generando otra matriz Y_t.

Una vez que se tiene la matriz Y tanto para los datos de entrenamiento como los de test, el siguiente paso consiste en realizar la evaluación utilizando KNN. Para ello, empleando las proyecciones de los datos de entrenamiento (matriz Y_e), se realizará una clasificación para cada proyección del conjunto de test (cada fila de la matriz Y_t), obteniendo n porcentajes de acierto distintos. La precisión de la transformación será evaluada por la media de los n porcentajes de acierto obtenidos.

Evaluación de la transformación lineal normalizada

Cabe destacar que las matrices Y_e y Y_t, resultantes de aplicar la transformación lineal pueden contener valores que son mayores que uno e incluso valores negativos, por lo que se realizará una normalización de dichas matrices para comprobar si esto mejora los resultados producidos en la reducción de la dimensionalidad.

Por tanto, se obtendrán dos nuevas matrices que denotaremos como Y_e' y Y_t' y serán el resultado de aplicar la normalización (con el procedimiento que se explicó en el apartado *2.1.2. Normalización*) a las matrices Y_e y Y_t, respectivamente.

Nuevamente, el resultado de esta transformación debe ser evaluado por KNN. Se utilizarán los datos de la matriz Y_e' para realizar la clasificación de los todos datos de test contenidos en la matriz Y_t', obteniendo así n porcentajes de acierto. Se calculará la tasa media de acierto de los n resultados obtenidos para evaluar el modelo.

Evaluación de la transformación realizada por la capa oculta, cuando se ha aplicado la función sigmoidal

El siguiente paso consistirá en aplicar la función de activación sigmoidal, tanto para la matriz Y_e como para la matriz Y_t. Es importante destacar que la función sigmoidal se aplica en las matrices Y_e y Y_t y no sobre las matrices normalizadas Y_e' y Y_t'.

La función sigmoidal debe ser aplicada a cada elemento de Y_e y Y_t, generando dos nuevas matrices S_e y S_t. Para realizar esto, se debe utilizar la siguiente ecuación:

$$s_{ij} = \frac{1}{1+e^{-y_{ij}}} \tag{5.2}$$

Donde:

- s_{ij} es el elemento de la fila *i* y columna *j* de la matriz *S*.
- y_{ij} es el elemento de la fila *i* y columna *j* de la matriz *Y*.

De forma análoga a los casos anteriores, se empleará la tasa media de acierto obtenida mediante KNN para evaluar la transformación. En este caso, se utilizará S_e para clasificar todos los datos de S_t.

En la *ilustración 5.2* se muestra de forma gráfica el proceso detallado a lo largo de este apartado. Como se comentó anteriormente, este proceso se repite en 50 ocasiones; habiendo fijado previamente el parámetro *k* para KNN, el número de neuronas en la capa oculta y el número de iteraciones de entrenamiento.

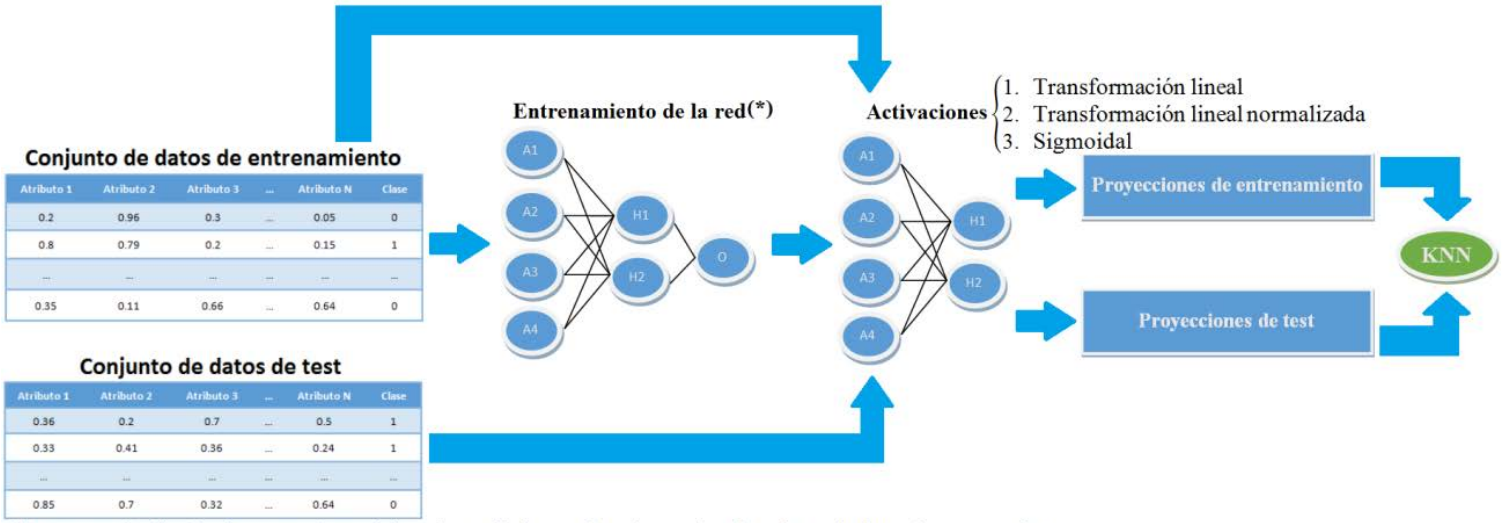

Ilustración 5.2: *Reducción de la dimensionalidad aplicando el MLP y evaluación mediante KNN*

En la *ilustración 5.3* se puede observar como se realizan las 50 repeticiones del mismo experimento. En primer lugar, se hacen 10 repeticiones, almacenando todos los resultados obtenidos en los siguientes casos:

- Tasa de acierto obtenida en la salida de la red realizando la clasificación de los datos de entrenamiento.
- Tasa de acierto obtenida por KNN realizando la clasificación de las proyecciones calculadas con la transformación lineal.
- Tasa de acierto obtenida por KNN realizando la clasificación de las proyecciones calculadas con la transformación lineal normalizada.

- Tasa de acierto obtenida por KNN realizando la clasificación de las proyecciones calculadas aplicando la función sigmoidal.

Una vez ejecutadas las 10 primeras repeticiones, se elige la mejor de ellas, para realizar esto se escoge la repetición que tenga la tasa de acierto más elevada sobre el conjunto de entrenamiento. De la mejor repetición se guardan las tasas de acierto de las tres versiones de reducción de dimensionalidad llevadas a cabo (transformación lineal, transformación lineal normalizada y sigmoidal). Este proceso se repite durante 5 veces, obteniendo así 5 ejecuciones. Por último, se calcula la tasa de acierto final para las tres aproximaciones como la media de las 5 ejecuciones obtenidas.

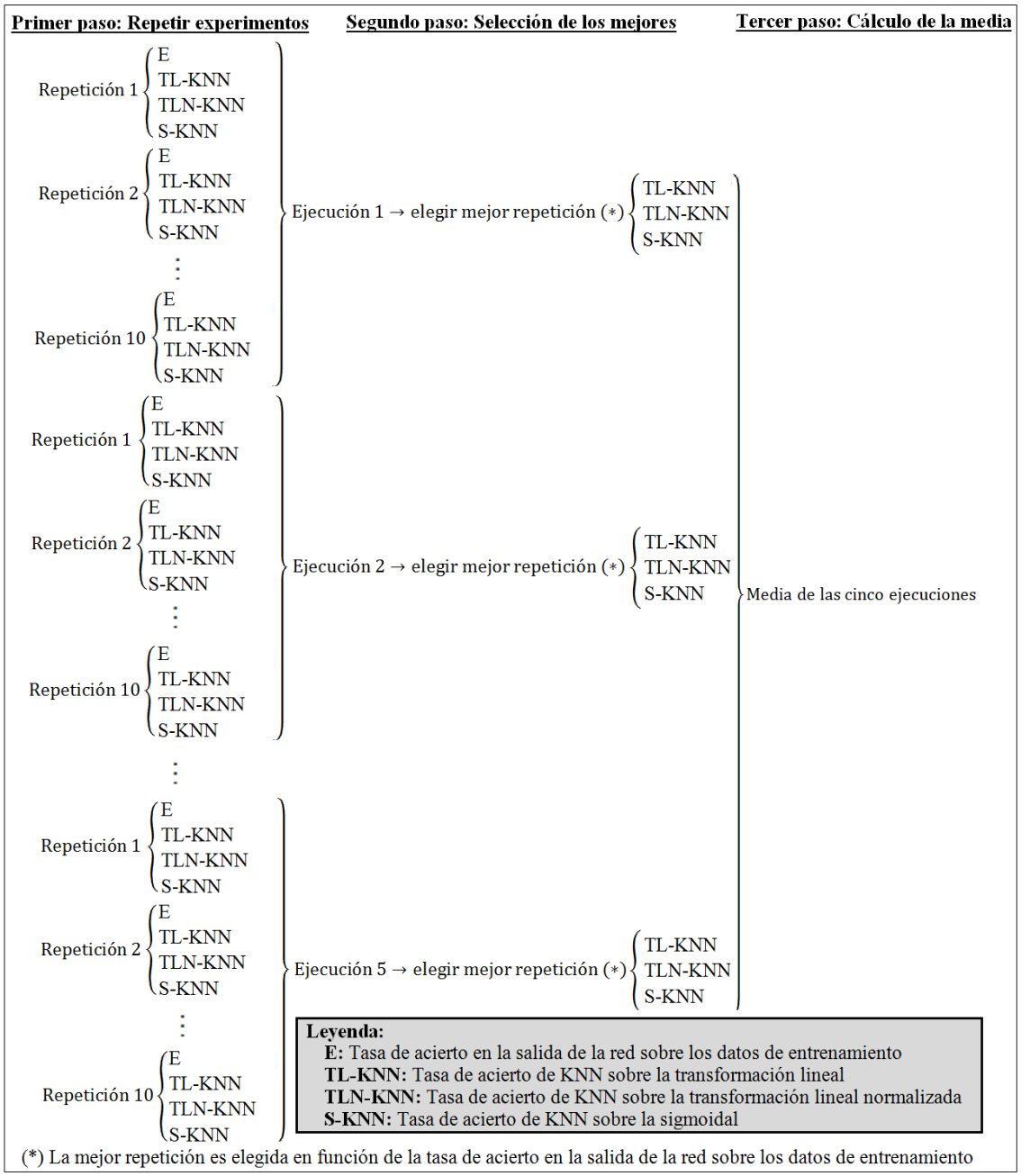

Ilustración 5.3: *Repetición de los experimentos, selección de los mejores y cálculo de la media*

5.1.2. Fase 2: Evaluación de la precisión de KNN sobre los datos originales

En cada dominio será medida la eficiencia del clasificador KNN aplicándolo sobre los datos originales. Para realizar esto, se utilizarán como muestras de entrenamiento el conjunto original de entrenamiento X_e y se realizará una clasificación para cada dato del conjunto de test original X_t, calculándose la tasa media de acierto para realizar la evaluación.

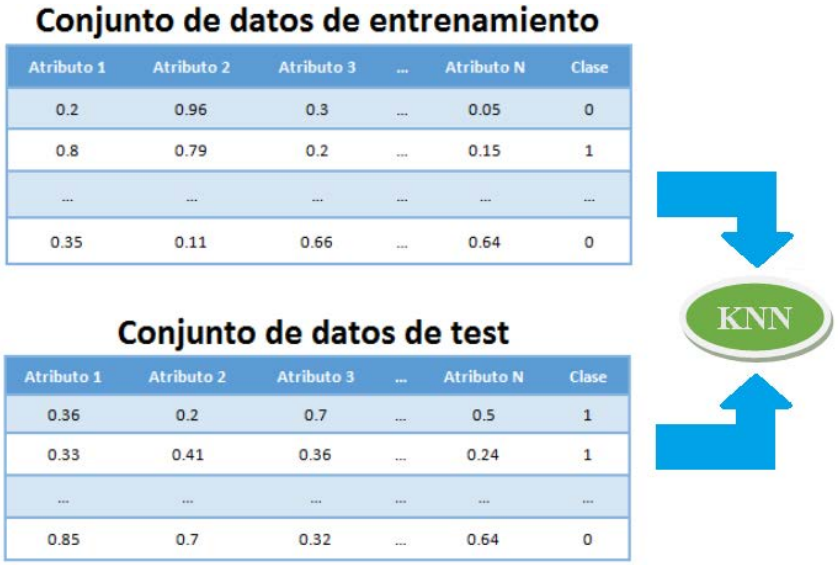

Ilustración 5.4: *Evaluación de la precisión de KNN sobre los datos originales*

5.1.3. Fase 3: Reducción de la dimensionalidad aplicando PCA y evaluación mediante KNN

En esta fase se utilizará PCA para llevar a cabo la reducción de la dimensionalidad. Por tanto, el primer paso consiste en el cálculo de las componentes principales utilizando el procedimiento descrito en el apartado *2.2.1. Cálculo de los componentes principales*. Es importante destacar que los componentes son calculados empleando únicamente la matriz datos de entrenamiento, X_e.

Una vez calculados todos los componentes principales, se eligen aquellos que más varianza explican de los datos y se forma la matriz U para realizar la transformación lineal. Nótese que, al igual que en el caso del Perceptrón Multicapa, serán realizados diferentes experimentos en cada dominio variando el número de componentes para encontrar cuál es el valor óptimo de componentes en cada caso.

El siguiente paso consiste en aplicar la transformación de los datos originales utilizando la matriz U. Para realizar esto, es necesario resolver la siguiente ecuación:

$$Y = UX^T \tag{5.3}$$

Donde:

- Y es la matriz con los datos con un número de dimensiones reducido.
- U es la matriz que contiene los k mejores componentes principales.
- X es la matriz que contiene los datos originales.

La ecuación anterior debe ser utilizada en dos ocasiones. En primer lugar, se realizará la transformación del conjunto de datos de entrenamiento X_e obteniendo la matriz de los datos de entrenamiento con dimensiones reducidas Y_e y, en segundo lugar, se realizará la transformación del conjunto de datos de test X_t obteniendo la nueva matriz de test que denotaremos como Y_t.

Por último, se mide la calidad de la transformación realizada por PCA empleando KNN, es decir, a partir de las proyecciones de los datos de entrenamiento (matriz Y_e) se hace una clasificación para cada dato del conjunto de test con dimensiones reducidas (cada fila de la matriz Y_t) y se calcula la tasa media de acierto.

En la siguiente ilustración podemos observar un gráfico que muestra el procedimiento para evaluar la reducción de dimensionalidad utilizando PCA.

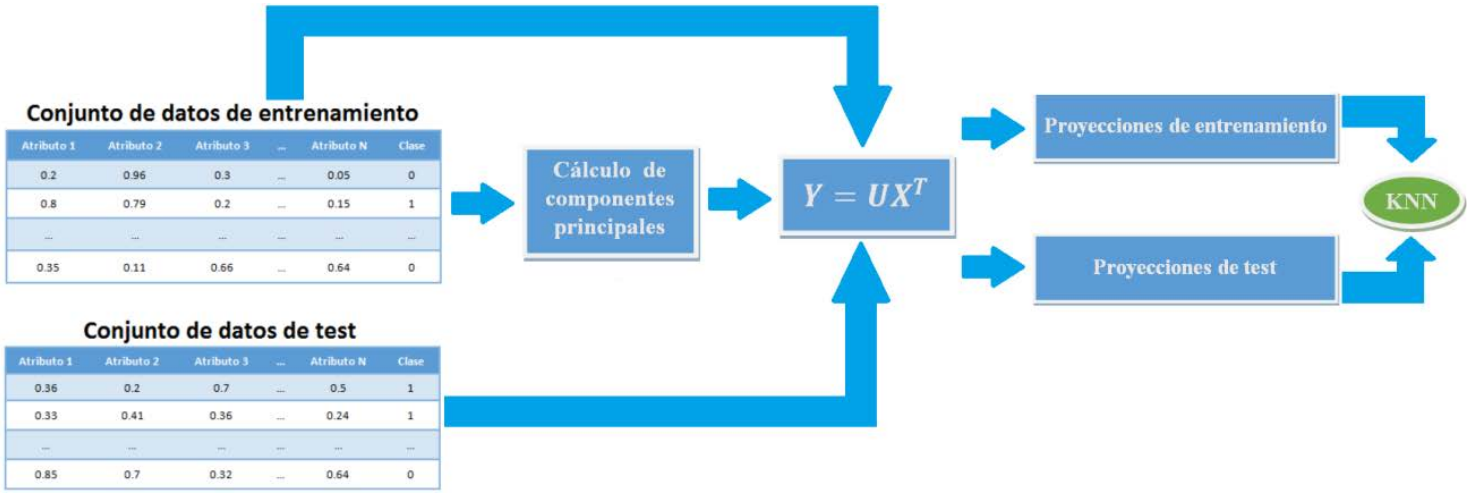

Ilustración 5.5: *Reducción de la dimensionalidad aplicando PCA y evaluación mediante KNN*

5.2. Alternativas de diseño

En el desarrollo del estudio se han tenido en cuenta diferentes opciones de diseño. Algunas de las alternativas barajadas han sido en los siguientes aspectos:

- Diseño de la red de neuronas.
- Repetición de experimentos.
- Número de neuronas en la capa oculta.
- Lenguaje de programación

Diseño de la red de neuronas

En cuanto se refiere al diseño de la red de neuronas se plantearon dos formas distintas para realizar la reducción de la dimensionalidad. El primer diseño planteado consistía en una arquitectura basada en un autocodificador con una sola capa oculta, es decir, la red neuronal tendría el mismo número de neuronas en la capa de entrada que atributos tuviese el conjunto de datos y, a su vez, este número sería igual al número de neuronas de la capa de salida. Sin embargo, la cantidad de neuronas de la capa oculta debería ser estrictamente menor que el número de neuronas de la capa de entrada y salida, ya que sus activaciones corresponderían con los atributos del nuevo conjunto de datos con una dimensión reducida. Esta primera aproximación se trataría de una reducción de la dimensionalidad no supervisada debido a que no se utiliza la salida esperada de los datos en el entrenamiento de la red.

El segundo diseño planteado fue el que definitivamente se decidió emplear y el que se explica detalladamente en anteriores apartados. La diferencia fundamental entre ambos planteamientos es que en el modelo adoptado la red neuronal tiene una única neurona en la capa de salida, donde se obtiene el valor esperado para cada patrón para realizar el entrenamiento. El modelo implementado se trata, por tanto, de un esquema supervisado.

El motivo fundamental por el que se escogió el esquema supervisado fue debido a que el objetivo general del trabajo es mejorar los resultados en problemas supervisados. Además, cabe destacar que el diseño final incorpora algunas ventajas en la fase de aprendizaje porque el proceso de entrenamiento de una arquitectura basada en un autocodificador es mucho más costoso ya que, al existir muchas más neuronas en la capa de salida es necesario ajustar más pesos en las conexiones de las neuronas de la capa de salida con las neuronas de la capa oculta.

Repetición de experimentos

En el diseño final de la solución, los experimentos de reducción de dimensionalidad realizados con el Perceptrón Multicapa son repetidos en 50 ocasiones y, posteriormente, la tasa de acierto es calculada como la media de 5 de estas repeticiones. Sin embargo, en el diseño inicial esto no fue planteado así.

En un primer momento los experimentos eran repetidos una única vez, pero se observó que a medida que aumentaba el número de neuronas en la capa oculta el rendimiento empezaba a empeorar y no se obtenían los resultados esperados. Esta situación era producida por la presencia de mínimos locales. Por ello, se decidió finalmente realizar los experimentos 50 veces y calcular la tasa de acierto como la media de 5 ejecuciones.

Número de capas ocultas de la red

También se barajó la posibilidad de incorporar más de una capa oculta en la red. Sin embargo, esta situación se descartó, ya que los resultados obtenidos únicamente con una capa oculta son bastante satisfactorios y entrenar este tipo de redes es costoso porque habitualmente se realiza un entrenamiento por capas.

Lenguaje de programación

Además del lenguaje R existen otro tipo de lenguajes que proporcionan facilidades para trabajar con el tipo de métodos empleados en el desarrollo del trabajo. Uno de estos lenguajes es Matlab, aunque este lenguaje de programación es muy similar a R no es de libre distribución por lo que se decidió utilizar R.

5.3. Requisitos

Una vez explicado el diseño del sistema, en este apartado se muestran los requisitos de usuario. Para la especificación de los requisitos se ha utilizado el estándar que se detalla en Métrica versión 3, a continuación se explican los campos que tendrá cada uno de los requisitos:

- *Identificación*: Cada requisito incluirá un identificador único. La codificación de los requisitos es UG-R*xx*, donde:
  - U: Indica que se trata de un requisito.
  - G: Indica que es un requisito general.

  - *xx*: Indica el código numérico que identifica a cada requisito de forma única.
- *Necesidad*: Los requisitos esenciales del usuario son no negociables; el resto pueden ser menos importantes y sujetos a la negociación.
- *Prioridad*: Cada requisito de usuario incluirá una medida de la prioridad que posee.
- *Estabilidad*: Algunos requisitos de usuario pueden permanecer estables durante toda la vida esperada del software; otros pueden ser más dependientes del feedback que se obtenga en los requisitos software, o el diseño.
- *Fuente*: Se indicará el origen de donde procede el requisito.
- *Claridad*: Un requisito de usuario es claro si tiene una, y solamente una, interpretación. La claridad implica la falta de ambigüedad. Si un término es utilizado en cierto contexto y tiene múltiples significados, se debe aclarar su interpretación o debe ser reemplazado por un término más específico.
- *Verificabilidad*: Cada requisito de usuario será verificable.

5.3.1. Requisitos funcionales

Identificador: UG-R01	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema realizará un preprocesado de los datos originales, concretamente se aplicarán las siguientes técnicas: • Normalización. • Aleatorización y separación de los datos en dos subconjuntos: El conjunto de entrenamiento tendrá el 60% del conjunto total de datos y el conjunto de test el resto.

Tabla 5.1: *Requisito funcional UG-R01*

Identificador: UG-R02	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema calculará en todos los experimentos la tasa media de acierto sobre el conjunto de datos de entrenamiento y registrará el resultado en un fichero de datos.

Tabla 5.2: *Requisito funcional UG-R02*

Identificador: UG-R03	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema obtendrá la transformación lineal que realiza la capa oculta de la red de neuronas.

***Tabla 5.3**: Requisito funcional UG-R03*

Identificador: UG-R04	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema normalizará la transformación lineal que realiza la capa oculta de la red de neuronas.

Tabla 5.4: *Requisito funcional UG-R04*

Identificador: UG-R05	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema aplicará la función sigmoidal al resultado producido por la transformación lineal de la capa oculta de la red.

Tabla 5.5: *Requisito funcional UG-R05*

Identificador: UG-R06	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema realizará el entrenamiento de la red de neuronas con el algoritmo de Retropropagación.

***Tabla 5.6:** Requisito funcional UG-R06*

Identificador: UG-R07	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema aplicará KNN para distintos valores de k (k = 1, 3, 5 y 7) y registrará el resultado producido en un fichero de datos en los siguientes casos: • Transformación lineal de la capa oculta. • Transformación lineal normalizada. • Función sigmoidal. • PCA.

Tabla 5.7: *Requisito funcional UG-R07*

Identificador: UG-R08	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema calculará las componentes principales con el conjunto de datos de entrenamiento y transformará los datos de entrenamiento y test por separado.

Tabla 5.8: *Requisito funcional UG-R08*

Identificador: UG-R09	
Prioridad: ☒ Alta ☐ Media ☐ Baja	Fuente: Álvaro
Necesidad: ☒ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☒ Alta ☐ Media ☐ Baja	Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El sistema repetirá los experimentos con el mismo número de neuronas en la capa oculta y componentes.

Tabla 5.9: *Requisito funcional UG-R09*

Identificador: UG-R10		
Prioridad: ☒ Alta ☐ Media ☐ Baja		Fuente: Álvaro
Necesidad: ☒ Esencial	☐ Deseable	☐ Opcional
Claridad: ☒ Alta ☐ Media ☐ Baja		Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema	
Descripción:	El sistema repetirá los experimentos realizados por la red de neuronas 50 veces, utilizando una inicialización de los pesos distinta en cada caso.	

Tabla 5.10: *Requisito funcional UG-R10*

Identificador: UG-R11		
Prioridad: ☒ Alta ☐ Media ☐ Baja		Fuente: Álvaro
Necesidad: ☒ Esencial	☐ Deseable	☐ Opcional
Claridad: ☒ Alta ☐ Media ☐ Baja		Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema	
Descripción:	El sistema agrupará las 50 repeticiones de cada experimento realizadas con la red de neuronas en grupos de 10, y elegirá la mejor repetición de cada uno de los grupos basándose en la tasa de acierto obtenida sobre el conjunto de datos de entrenamiento, adquiriendo las 5 mejores repeticiones.	

Tabla 5.11: *Requisito funcional UG-R11*

Identificador: UG-R12		
Prioridad: ☒ Alta ☐ Media ☐ Baja		Fuente: Álvaro
Necesidad: ☒ Esencial	☐ Deseable	☐ Opcional
Claridad: ☒ Alta ☐ Media ☐ Baja		Verificabilidad: ☒ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema	
Descripción:	El sistema calculará la media de las 5 mejores repeticiones para obtener la tasa de acierto en los experimentos realizados sobre la red de neuronas.	

Tabla 5.12: *Requisito funcional UG-R12*

5.4. Código implementado

En este apartado se describe el código final implementado para llevar a cabo la experimentación y, posteriormente, con los resultados obtenidos realizar el estudio. Como se comentó anteriormente, el código ha sido implementado en el lenguaje R.

5.4.1. Script para realizar los experimentos con la red de neuronas

En este script se realizan los experimentos relativos a la reducción de la dimensionalidad con la red de neuronas. A continuación, se explica detalladamente el funcionamiento del código implementado.

Función *normalizarDatos*

La primera función implementada se llama *normalizarDatos* y se encarga de realizar la normalización de los datos en el intervalo [0,1], recibe por parámetro una columna de un conjunto de datos. Esta función realiza una comprobación para asegurar que los datos recibidos son de tipo numérico, en caso de serlo se aplica la fórmula de normalizar los datos (véase apartado *2.1.2. Normalización*), calculando el valor máximo con la función *max* de la librería *base* y el valor mínimo con la función *min* de la misma librería. Por último, la función devuelve la columna con los valores normalizados.

Función *aleatorizarDatos*

La función *aleatorizarDatos* se encarga de realizar la aleatorización de los datos y separarlos en el conjunto de entrenamiento y de test. Recibe por parámetro un data frame que contiene el conjunto de datos a aleatorizar y el porcentaje de datos que se desea que tenga el conjunto de datos de entrenamiento.

Esta función crea un array con datos aleatorios entre 1 y el número total de instancias. Estos números aleatorios son utilizados como índices para acceder de forma desordenada al data frame que contiene los datos y almacenarlos con una ordenación distinta en otro data frame llamado *rd*. Posteriormente, se crea otro array con valores aleatorios con el mismo rango que en el caso anterior, sin embargo en este caso se obtienen únicamente tantos números aleatorios como instancias debe tener el conjunto de entrenamiento (el 60% de la cantidad de datos total). Nuevamente, estos valores aleatorios son utilizados como índices para acceder de forma desordenada a los datos, generando así el conjunto de datos de entrenamiento y el conjunto de datos de test con las instancias que no son utilizadas para entrenamiento.

En este caso se retorna una lista que contiene dos data frames, el conjunto de datos de entrenamiento y test.

Función *nnp*

La función *nnp* se encarga de realizar el entrenamiento de la red de neuronas, utilizando el algoritmo de Retropagación. Esta función recibe como parámetro un data frame con el conjunto de datos de entrenamiento, un data frame con el conjunto de datos de test, el número de neuronas ocultas de la red y el número de iteraciones.

En primer lugar, la función realiza el entrenamiento de la red utilizando la función *nnet* de la librería *nnet*. Posteriormente, se obtienen los pesos que son calculados en el proceso de entrenamiento por la función anterior. Los pesos de las conexiones que unen las neuronas de la capa de entrada con la capa oculta son almacenados en un array con forma de matriz llamado *matrix1*, mientras que los pesos que unen las neuronas de la capa oculta con la capa de salida son almacenados en otro array llamado *matrix2*.

Por último, se retorna una lista que contiene una función con el nombre *projection* que permite calcular la transformación lineal de la capa oculta de la red, una estructura

de datos llamada *mod* que contiene toda la información de la red y, además, se incluyen las matrices con los pesos *matrix1* y *matrix2*.

Función *entrenarRed*

La función *entrenarRed* recibe como parámetro un data frame que contiene el conjunto de datos de entrenamiento, otro data frame que contiene el conjunto de datos de test, el número de neuronas que debe tener la capa oculta, el número de iteraciones con el que será realizado el entrenamiento y la semilla que será utilizada para obtener las inicializaciones de los pesos.

Esta función se encarga de establecer la semilla que será utilizada para inicializar los pesos de la red de neuronas. Después, se llama a la función *nnp* (función descrita anteriormente) para realizar el entrenamiento de la red, guardando la lista que devuelve esta función para después retornarla.

Por tanto, esta función retorna una lista que contiene una función llamada *projection*, una estructura de datos llamada *mod* que contiene información de la red de neuronas, una matriz llamada *matrix1* que contiene los pesos de las conexiones entre la capa de entrada y la capa oculta y otra matriz llamada *matrix2* que contiene los pesos de las conexiones entre la capa oculta y la capa de salida.

Función *nn*

La función *nn* recibe como parámetro un data frame con el conjunto de datos de entrenamiento, un data frame con el conjunto de datos de test, la información de la red contenida en la estructura de datos llamada *mod*, el número de repetición del experimento, el número de ejecución del experimento, el número de iteraciones con el que ha sido entrenada la red de neuronas, el nombre del fichero donde serán registrados los resultados y la semilla con la que ha sido realizada la inicialización de los pesos.

Esta función lo primero que hace es calcular la salida de la red para cada dato del conjunto de entrenamiento y guardarlo en la variable *outputstrain*. Cabe destacar que la salida de la red es un valor continuo, por este motivo, cuando se tienen los valores de salida para los datos del conjunto de entrenamiento (*outputstrain*), el siguiente paso que se realiza es interpretar estos valores de la siguiente forma:

- Si la salida calculada es superior o igual a 0,5, el resultado se interpreta como la primera clase.
- Si la salida calculada es inferior a 0,5, el resultado se interpreta como la segunda clase

Los valores interpretados se vuelven a almacenar en la variable *outputstrain*, sobreescribiendo los valores anteriores. Después, la función calcula la tasa de acierto, para realizar esto compara los resultados almacenados en *outputstrain* con la clase real de cada dato y realiza la división de los datos correctamente clasificados entre el número total de ejemplos. Por último, los siguientes datos se registran en el fichero con el nombre que se recibe como parámetro:

- Se almacena la palabra *train*. Esto se hace para saber posteriormente que el resultado de clasificación es obtenido sobre el conjunto de datos de entrenamiento.
- El número de repetición del experimento.
- El número de ejecución del experimento.

- Se almacena que la tasa de acierto corresponde con la salida en la red de neuronas. Esto se realiza añadiendo la palabra *nnet* en la cadena de texto.
- El número de neuronas de la capa oculta con el que se ha realizado el experimento.
- El número de iteraciones con el que ha sido entrenada la red.
- Se almacena un número 0 en todos los casos. Esto se hace para facilitar el análisis de los datos y mantener el mismo formato que utiliza la función que se describe a continuación (*knn.nn*)
- Tasa de acierto obtenida.
- La semilla con la que se han inicializado los pesos.

A continuación, se muestra un ejemplo almacenado en un fichero con estos campos:

```
train  1 1 nnet 4 100 0 0.805 301
```

Función *knn.nn*

La función *knn.nn* recibe como parámetro un data frame con el conjunto de datos de entrenamiento, otro data frame que contiene el conjunto de datos de test, una función llamada *projection*, el número de neuronas en la capa oculta, el número de iteraciones con el que se ha realizado el entrenamiento, el número de ejecución del experimento, el número de repetición del experimento, el parámetro *k* o número de vecinos, el nombre del fichero donde se deben registrar los datos y la semilla con la que se han inicializado los pesos.

Esta función calcula la transformación lineal utilizando la función llamada *projection* que se recibe como parámetro. El resultado de las proyecciones del conjunto de entrenamiento se almacena en *projected_train*, mientras que las proyecciones del conjunto de test se almacenan en *projected_test*.

Una vez calculadas ambas proyecciones, se utiliza la función llamada *knn* de la librería *FNN* para hacer la clasificación de las proyecciones del conjunto de test (*projected_test*) utilizando las proyecciones del conjunto de entrenamiento (*projected_train*). Además, a la función *knn* también se le proporciona el parámetro *k*. El siguiente paso consiste en obtener la tasa de acierto de la clasificación, para ello se comparan los resultados de clasificación obtenidos por la función *knn* con la clase real de los datos, contabilizando los aciertos que se producen, y se realiza la división del número de aciertos y el número total de instancias. Posteriormente, se guarda este resultado en el fichero de texto.

Después se realiza la normalización de los resultados obtenidos en la transformación lineal utilizando la función *normalizarDatos* (descrita anteriormente). De forma análoga al caso anterior, se llama a la función *knn* para realizar la clasificación sobre las nuevas proyecciones de test y se calcula la tasa de acierto. Posteriormente, se registran los resultados en el fichero correspondiente.

Por último, se aplica la función *sigmoid* de la librería *pracma* sobre las variables *projected_train* y *projected_test*. También, se utiliza la función *knn* para realizar la clasificación, se calcula la tasa de acierto obtenida y se registran los resultados en el fichero.

Es importante destacar que esta función guarda la siguiente información en el fichero:

- Se incluye la palabra *test* porque en este caso la tasa de acierto obtenida es siempre sobre los datos de test. Esto se realiza para facilitar el posterior análisis de los datos.
- El número de repetición del experimento.
- El número de ejecución del experimento.
- Se almacena a la versión que pertenece la tasa de acierto que se registra. Este parámetro puede tomar los siguientes valores: *Linear*, *LinearNorm* y *Sigmoid*.
- El número de neuronas de la capa oculta con el que se ha realizado el experimento.
- El número de iteraciones con el que ha sido entrenada la red.
- El parámetro *k* que utiliza KNN para realizar la clasificación.
- Tasa de acierto obtenida.
- La semilla con la que se han inicializado los pesos.

En las siguientes líneas se muestran algunos ejemplos de cada caso:

```
test 1 1 Linear 4 100 1 0.7725 301
test 1 1 LinearNorm 4 100 1 0.715 301
test 1 1 Sigmoid 4 100 1 0.66 301
```

Cuerpo principal del programa

El primer paso consiste en leer los datos con la función *data* de la librería *utils*, después se normalizan los datos con la función *normalizarDatos* y realiza una aleatorización y separación de los datos con la función *aleatorizarDatos*.

Posteriormente, se realiza un cálculo para asegurar que no se repita la semilla en ningún caso y, por tanto, asegurar de esta forma que la red de neuronas es inicializada con unos pesos diferentes en cada repetición de los distintos experimentos.

Para realizar los experimentos se utilizan cuatro bucles anidados, el primero de ellos se encarga de recorrer los distintos números de neuronas ocultas utilizados, el segundo bucle se ejecuta dos veces (100 y 1.000 iteraciones) con el número de iteraciones que se utiliza para entrenar la red, el tercero se ejecuta siempre 10 veces y el último bucle se ejecuta 5 veces. Estos dos últimos bucles se utilizan para repetir los experimentos 50 veces, el motivo por el que se utilizan dos bucles es para facilitar el análisis posterior de los resultados. Dentro del cuerpo de estos bucles anidados, en primer lugar, se realiza el entrenamiento de la red con la función *entrenarRed* y se llama a la función *nn* para registrar tasa de acierto en la salida de la red. Por último, se utiliza otro bucle para llamar a la función *knn.nn* en 4 ocasiones con distintos valores del parámetro *k*.

5.4.2. Script para realizar los experimentos con PCA y aplicar KNN sobre los datos originales

En este script se aplica KNN sobre los datos originales y se realizan los experimentos relativos a la reducción de la dimensionalidad con PCA. A continuación, se explica detalladamente el funcionamiento del código implementado.

Función *knn.*

La función *knn.* recibe como parámetro un data frame con el conjunto de datos de entrenamiento, un data frame con el conjunto de datos de test y el parámetro *k.*

Esta función hace una llamada a la función *knn* de la librería *FNN* que retorna la clase esperada de cada dato de test. Después, la función contabiliza los datos de test que han sido clasificados de forma correcta, es decir, aquellos en los que coincide el valor esperado obtenido en la clasificación con el real. Por último, para calcular la tasa de acierto se realiza la división entre el número de aciertos obtenidos y el número total de datos existentes. La función retorna esta tasa de acierto.

Función *pca.knn*

La función *pca.knn* recibe como parámetro un data frame con el conjunto de datos de entrenamiento, otro data frame con el conjunto de datos de test, el parámetro *k* y el número de componentes para PCA.

En primer lugar, la función calcula las componentes principales con el conjunto de datos de entrenamiento. Para realizar esto, se utiliza la función *preprocess* de la librería *caret*. Posteriormente, se transforman los datos con la función *predict* de la librería *caret*, la transformación de los datos de entrenamiento y test se realiza por separado almacenándose los resultados en las variables *pcatrain* y *pcatest*, respectivamente. Por último, se utiliza la función *knn.* (explicada anteriormente) para obtener y retornar la tasa de acierto sobre las proyecciones de los datos de test.

Función *aleatorizarDatos*

Esta función es la misma que la función *aleatorizarDatos* del script anterior.

Función *guardarFichero*

La función *guardarFichero* recibe como parámetro una cadena de texto y el nombre del fichero donde se guardan los datos. Esta función únicamente guarda la cadena de texto en el fichero cuyo nombre es indicando por parámetro.

Por un lado, el formato que se utiliza para almacenar los resultados de KNN sobre los datos originales es el siguiente:

- El parámetro *k* que utiliza KNN para realizar la clasificación.
- La tasa de acierto obtenida.

En la siguiente línea se muestra un ejemplo que contiene un resultado con los campos descritos anteriormente:

```
1 0.673075
```

Por otro lado, el formato utilizado para registrar los datos de PCA en el fichero de texto contiene los siguientes campos:

- El parámetro *k* que utiliza KNN para realizar la clasificación.
- El número de componentes utilizados en PCA.
- La tasa de acierto obtenida.

A continuación, se muestra un ejemplo con los campos almacenados en el fichero de texto para los resultados de PCA:

```
1  1 0.528425
```

Cuerpo principal del programa

Con un bucle se recorren los distintos valores de *k* utilizados con KNN. Dentro de este bucle se llama a la función *knn.* para obtener el porcentaje de acierto en la clasificación de los datos de test originales, es decir, sin reducir su dimensionalidad. Adicionalmente, existe otro bucle anidado para llamar a la función *pca.knn* con distinto número de componentes. Cabe destacar que los resultados obtenidos en la clasificación de los datos originales y las proyecciones de PCA son almacenados en un fichero con la función *guardarFichero.*

5.4.3. Script para procesar los resultados

Un tercer script ha sido creado, utilizando la librería *dplyr*, para procesar de forma automática los resultados obtenidos con la reducción de la dimensionalidad realizada con la red de neuronas.

Este script primero carga el fichero de texto resultante de la ejecución del script explicado en el apartado *5.4.1. Script para realizar los experimentos con la red de neuronas*. Para realizar la carga de datos se utiliza la función *read.table* de la librería *utils*.

Como se comentó anteriormente, la elección de los mejores experimentos es realizada en función de la tasa de acierto obtenida en la salida de la red sobre los datos de entrenamiento. Por este motivo, con la herramienta *dplyr* se filtran los resultados, eligiendo los de entrenamiento. Después, estos resultados se agrupan en grupos de 10 para elegir el mejor experimento de cada uno de ellos y, por tanto, obtener los 5 experimentos que menos error presentan en el conjunto de entrenamiento. Para coger los resultados de test asociados a los mejores experimentos seleccionados se filtran los experimentos por la semilla que es única en cada ejecución.

Posteriormente, se calcula la media de la tasa de acierto en la salida de la red sobre los datos de test de los 5 mejores experimentos encontrados anteriormente. El cálculo se repite con los distintos números de neuronas en la capa oculta utilizadas y con las 100 y 1.000 iteraciones. Estos datos se guardan en un fichero y se crea un gráfico con la herramienta *ggplot*. Además, cabe destacar que esto se hace también con la transformación lineal, la transformación lineal normalizada y la sigmoidal, pero en estos casos el proceso además se realiza con los distintos valores de *k* utilizados.

5.4.4. Integración de un nuevo modelo para *Caret*

Durante el desarrollo del trabajo se propuso como objetivo extra la creación de una librería para que el método de reducción de dimensionalidad estudiado pudiese ser usado por la comunidad. Como se verá en el capítulo 6, los resultados obtenidos de reducción de dimensionalidad se consideran muy buenos, por lo que el objetivo de crear el paquete finalmente se decidió llevar a cabo. Por tanto, en este apartado se describe el código de esta nueva librería, así como ejemplos de su uso.

5.4.4.1. Librería *Caret*

En lugar de crear un nuevo paquete, la decisión final fue la de integrar el método de reducción de dimensionalidad en una librería existente y muy conocida como es *Caret*. Esta librería, implementada en el lenguaje de programación R, contiene un conjunto de

funciones que intentan agilizar el proceso para crear modelos predictivos, es decir, contiene herramientas para hacer las siguientes tareas:

- División de datos.
- Preprocesado de datos.
- Selección de características.
- Ajuste de modelo mediante remuestreo.
- Estimación de la importancia de las variables.

Caret se inició con el objetivo de crear una interfaz uniforme de las funciones en sí, así como una forma de estandarizar las tareas comunes. La actual versión puede ser encontrada en el repositorio conocido con el nombre *CRAN* y el código fuente del proyecto está almacenado en GitHub.

En esta librería es posible añadir nuevos modelos, pero no existe la posibilidad de crear nuevos métodos de preprocesado de datos. Por tanto, se tomó la decisión de realizar una fase inicial de reducción de dimensionalidad y, posteriormente, utilizar KNN para realizar la clasificación de los datos.

Para añadir un nuevo modelo a Caret es necesario crear una lista de componentes con un nombre previamente establecido. Esta lista es usada por una función, conocida con el nombre de *train*, que es la que realiza todos los experimentos de forma totalmente automática. A continuación, se describen algunas de los componentes más importantes que utiliza dicha función:

- *Label*: Es una cadena de caracteres que permite definir una descripción de la librería que se implementa.
- *Library*: Es un array de caracteres que contiene el nombre de las librerías externas que se utilizan.
- *Type*: Es una array de caracteres que sirve para indicar si se trata de un modelo de clasificación o regresión. Si es clasificación, debe indicarse con la palabra *Classification*; mientras que si es regresión, se indica con la palabra *Regression*. Cabe destacar que también puede tomar ambos valores.
- *Parameters*: Es un data frame que tiene tres valores que indican los parámetros que deben tenerse en cuenta a la hora de realizar el ajuste. Los valores que es necesario definir para cada parámetro son: el nombre, el tipo de dato y una etiqueta textual para definir el parámetro.
- *Grid*: Es una función que se usa para definir todos los parámetros con los que queremos hacer pruebas.
- *Fit*: Es una función que es utilizada para ajustar el modelo. También es posible realizar técnicas de preprocesado de datos en esta función.
- *Predict*: Es una función que se utiliza para realizar las predicciones, es decir, calcula los valores estimados para un conjunto de datos.

5.4.4.2. Modelo implementado

Teniendo en cuenta las restricciones que define Caret para añadir un nuevo modelo, se ha creado un código con una lista que contiene los componentes descritos anteriormente. A continuación, se realiza una explicación del código implementado.

Componente *label*

En este caso se ha definido la librería con la siguiente cadena de caracteres: *KNN with Feature Extraction*.

Componente *library*

Cabe destacar que el código no funcionará sin haber instalado previamente las librerías que aquí se indican, por lo que es necesario que estén instaladas previamente. Las librerías externas utilizadas han sido las siguientes:

- *nnet*: Con esta librería se hace el entrenamiento de la red mediante el algoritmo de retropropagación.
- *FNN*: Con esta librería se aplica KNN a los datos.
- *NeuralNetTools*: Con esta librería se obtienen los pesos de las conexiones de las neuronas cuando se ha realizado el entrenamiento.

Componente *type*

Este componente contiene únicamente la palabra clave *Classification*, ya que nuestro modelo se trata de clasificación.

Componente *parameters*

En este caso es necesario el ajuste de tres parámetros que son: el número de neuronas en la capa oculta de la red, el número de iteraciones de la red y el parámetro *k* de KNN.

Componente *grid*

En esta función han sido implementadas dos formas para obtener los parámetros con los que van a ser realizados los experimentos. En primer lugar, se propone una búsqueda exhaustiva del número de neuronas óptimo que debe tener la red de neuronas en la capa oculta de la red, es decir, se busca desde una neurona hasta el número de dimensiones máximo que se indica por parámetro. En segundo lugar, se propone una búsqueda aleatoria, realizando un total de 4 arquitecturas de la red con un número distinto de neuronas en la capa oculta. Además, cabe destacar que en ambos casos se realiza el entrenamiento con 100 y 1.000 iteraciones y el parámetro *k* que se utiliza es 1, 3, 5 y 7.

Componente *fit*

En primer lugar, esta función es utilizada para realizar el entrenamiento de la red de neuronas utilizando la función *nnet* de la librería *nnet*. Posteriormente, se obtienen los pesos utilizando la librería *NeuralNetTools*. Por último, se realizan las proyecciones de los datos de entrenamiento. Esta función devuelve una lista que contiene:

- Una estructura de datos, llamada *modelNN*, para almacenar toda la información de la red de neuronas.
- Una lista, llamada *modelKNN*, que contiene las proyecciones de los datos de entrenamiento y las clases de estas proyecciones.
- El parámetro *k*.

Componente *predict*

Esta función utiliza la información de la estructura de datos llamada *modelNN* (estructura de datos que se retorna en la función *fit*) para obtener los pesos de las conexiones de las neuronas de la primera capa y la capa oculta, y poder calcular así las activaciones de los datos de test. Una vez que son calculadas dichas proyecciones, se

obtienen los resultados de clasificación, aplicando KNN con la librería *FNN*. Esta función retorna los resultados de clasificación obtenidos por KNN.

5.4.4.3. Manual de usuario

Las funciones anteriormente descritas no son utilizadas de forma directa, ya que son llamadas desde la función *train*. Esta función nos indicará de forma automática cuál son los parámetros (el número de neuronas en la capa oculta, número de iteraciones de entrenamiento y parámetro *k*) que debemos utilizar en nuestro dominio para llevar a cabo la reducción de la dimensionalidad de forma óptima.

A la función *train* se le deben pasar una serie de parámetros. Los parámetros más destacables son los siguientes:

- *x*: Es un objeto donde los ejemplos son las filas y los atributos corresponden con las columnas. Puede ser una matriz, un data frame u otro tipo, pero debe tener nombre de columnas.
- *y*: Es una lista numérica que contiene los resultados para cada ejemplo.
- *method*: Especifica el nombre del modelo a utilizar. En este caso el nombre será *nnproj*.
- *preProcess*: Un array que define técnicas de preprocesado de datos para ser aplicadas. Los actuales valores que pueden tomar son: *BoxCox*, *YeoJohnson*, *expoTrans*, *center*, *scale*, *range*, *knnImpute*, *bagImpute*, *medianImpute*, *pca*, *ica* y *spatialSign*.
- *tuneGrid*: Es un data frame que contiene los parámetros que serán utilizados en los experimentos.
- *trControl*: Es una lista de valores que permiten controlar el proceso de aprendizaje. Una de estas opciones, llamada *search*, permite seleccionar una de las dos opciones de búsqueda implementada en la función *grid*. Para una búsqueda exhaustiva indicaremos la palabra *grid*, mientras que para una búsqueda aleatoria indicaremos la palabra *random*.

A continuación, se muestran algunos ejemplos de ejecución empleando datos del dominio *Spambase* (la descripción de este dominio se hace en el apardado *6.3. Dominio Spambase*). En la siguiente imagen se muestra el primer ejemplo de ejecución:

```
e <- train(spambase[-ncol(spambase)],as.factor(spambase[[ncol(spambase)]]),method=nnproj,trControl=trainControl(search="random"))
```

Ilustración 5.6: *Primer ejemplo de ejecución*

En este caso los parámetros que se le indican a la función son los siguientes:

- Spambase[-ncol(spambase)]: Corresponde con los datos del dominio Spambase.
- Spambase[[ncol(spambase)]]: Corresponde con las clases del conjunto de datos.
- TrainControl(search = “random”): Se indica que se desea realizar una búsqueda aleatoria del número de neuronas de la capa oculta.

El resultado de la ejecución anterior es el que podemos observar en la *ilustración 5.7*, donde se muestran todos los resultados de los distintos experimentos que se han realizado de forma automática. Por último, se muestra el valor óptimo de los parámetros encontrados (número de neuronas en la capa oculta, número de iteraciones y parámetro *k*). También se puede observar más información sobre el proceso de entrenamiento que se ha llevado a cabo, así por ejemplo es posible observar que la técnica de evaluación

empleada ha sido *bootstrapped* (esta técnica es explicada en el apartado *2.3.3. Técnicas de evaluación*).

```
KNN with Feature Extraction

4601 samples
  57 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 4601, 4601, 4601, 4601, 4601, 4601, ...
Resampling results across tuning parameters:

  hidden  steps  k  Accuracy   Kappa
   8       100   1  0.7879531  0.5608552
   8       100   3  0.7794426  0.5425663
   8       100   5  0.7878368  0.5585609
   8       100   7  0.7851065  0.5519029
   8      1000   1  0.7864566  0.5574353
   8      1000   3  0.7768650  0.5376602
   8      1000   5  0.7800346  0.5421984
   8      1000   7  0.7843461  0.5514015
  18       100   1  0.8099381  0.6058744
  18       100   3  0.8035271  0.5926390
  18       100   5  0.8006415  0.5858938
  18       100   7  0.7954319  0.5741947
  18      1000   1  0.8078631  0.6016315
  18      1000   3  0.7967305  0.5784283
  18      1000   5  0.7945089  0.5733337
  18      1000   7  0.7936657  0.5700002
  26       100   1  0.8302588  0.6474563
  26       100   3  0.8173604  0.6203323
  26       100   5  0.8176932  0.6208623
  26       100   7  0.8261318  0.6377625
  26      1000   1  0.8282851  0.6439382
  26      1000   3  0.8184937  0.6230098
  26      1000   5  0.8136206  0.6122129
  26      1000   7  0.8174767  0.6193347
  27       100   1  0.8338117  0.6546390
  27       100   3  0.8238914  0.6341503
  27       100   5  0.8232218  0.6318777
  27       100   7  0.8203492  0.6256639
  27      1000   1  0.8377924  0.6630217
  27      1000   3  0.8198527  0.6260605
  27      1000   5  0.8148571  0.6146835
  27      1000   7  0.8169022  0.6185088

Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were hidden = 27, steps = 1000 and k = 1.
```

Ilustración 5.7: *Resultado de la ejecución del primer ejemplo*

Cuando aumentan el número de experimentos es posible que sea complejo interpretar la información que se muestra como resultado de la ejecución. Por ello, es posible visualizar los resultados de la mejor ejecución utilizando la función *getTrainPerf*, también es posible observar los parámetros óptimos que se han obtenido ejecutando *nombre$bestTune.*

```
> getTrainPerf(e)
  TrainAccuracy TrainKappa method
1     0.8377924  0.6630217 custom

> e$bestTune
   hidden steps k
29     27  1000 1
```

Ilustración 5.8: *Visualicación de información complementaria*

Por otro lado, se ha hecho un ejemplo de ejecución adicional utilizando un número personalizado de ejecuciones. Para realizar esto, se ha creado un data frame que contiene los valores que deben tomar dichos parámetros.

```
> param
  hidden steps k
1      3   100 1
2      3   100 3
3      4   100 1
4      4   100 3
```

Ilustración 5.9: *Data frame con los parámetros*

El data frame anterior es introducido como parámetro a la función *train* mediante *tuneGrid*. Para realizar esto, se debe hacer la siguiente ejecución:

```
e <- train(spambase[-ncol(spambase)],as.factor(spambase[[ncol(spambase)]]),method = nnproj,tuneGrid = param)
```

Ilustración 5.10: *Segundo ejemplo de ejecución*

En la siguiente ilustración se observa el resultado de la ejecución.

```
KNN with Feature Extraction

4601 samples
  57 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 4601, 4601, 4601, 4601, 4601, 4601, ...
Resampling results across tuning parameters:

  hidden  k  Accuracy   Kappa
  3       1  0.7463193  0.4740385
  3       3  0.7498574  0.4797277
  4       1  0.7663194  0.5154824
  4       3  0.7539668  0.4890482

Tuning parameter 'steps' was held constant at a value of 100
Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were hidden = 4, steps = 100 and k = 1.
```

Ilustración 5.11: *Resultado de ejecución del segundo ejemplo*

Adicionalmente, es posible mostrar gráficos de los resultados obtenidos en los experimentos utilizando la función *ggplot*. A continuación, se muestra el gráfico obtenido de la ejecución anterior.

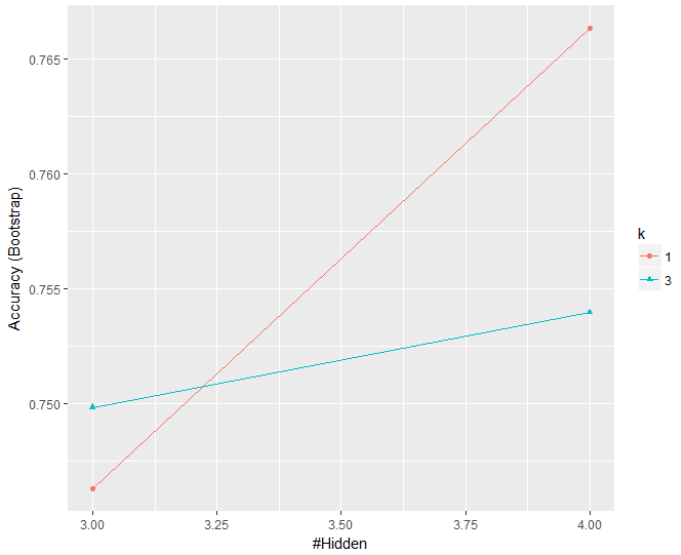

Ilustración 5.12: *Gráfica generada con la herramienta ggplot*

Capítulo 6

Estudio realizado

6.1. Introducción

Como se ha comentado anteriormente, el estudio de la viabilidad del esquema propuesto será realizado de forma empírica, por lo que se utilizarán cuatro conjuntos de datos para realizar la experimentación. Los dominios elegidos son los siguientes:

- Dominio *Doughnut*
- Dominio *Spambase*
- Dominio *German*
- Dominio *Splice*

El dominio conocido con el nombre de *doughnut* ha sido generado de forma artificial, mientras que los otros tres dominios han sido obtenidos del repositorio de datos UCI.

A lo largo de este capítulo se realiza una descripción más detallada de cada uno de estos conjuntos de datos y se muestran los resultados obtenidos para los experimentos llevados a cabo. Además, en cada conjunto de datos se muestra un análisis de los resultados obtenidos. En este análisis se comprueba si el resultado de clasificación obtenido sobre el conjunto de datos original es mejorado, clasificando el conjunto de datos con dimensiones reducidas. Además, se realiza la comparación de los resultados obtenidos con la red de neuronas y PCA.

La comparación con PCA es realizada obteniendo una tabla, cuyos valores son calculados mediante la resta de los resultados obtenidos con la red de neuronas y PCA. Por ello, la mejora obtenida con la red de neuronas será mayor cuanto mayor sean los resultados de la tabla y, por el contrario, valores negativos indican que se pierde precisión con respecto a PCA. Adicionalmente, con el fin de complementar el análisis comparativo, en cada dominio se muestra una gráfica que contiene los resultados de la transformación lineal, habiendo entrenado la red con 1.000 iteraciones, y PCA. Con la transformación lineal normalizada y la sigmoidal no se harán gráficas, ya que sería necesaria la creación de gran cantidad de gráficas y los resultados de las tres versiones no varían de una forma amplia.

Cabe destacar que el análisis anterior será realizado por separado en las tres versiones de reducción de dimensionalidad llevados a cabo por la red de neuronas, es decir, con la transformación lineal, la transformación lineal normalizada y aplicando la función sigmoidal (el procedimiento de obtención de estas tres versiones es explicado con detalle en el apartado *5.1.1.4. Evaluación de las activaciones de las neuronas de la capa oculta*).

En los dominios reales es frecuente encontrar atributos que no aportan nada en la resolución del problema e incluso pueden provocar que se obtengan peores resultados. Para simular esta situación y comprobar si la red de neuronas es capaz de extraer la información relevante de cada problema, en cada dominio se realizarán experimentos añadiendo atributos adicionales cuyos valores serán calculados de forma totalmente aleatoria. Adicionalmente, en algunos casos se realizará una rotación de los datos para distribuir el ruido de los atributos aleatorios entre todos los nuevos atributos. Si no se hace la rotación, existirán los atributos originales con información útil y los atributos añadidos con información irrelevante o ruido. Al hacer la rotación, se mezcla todo, y existirá ruido en todos los atributos.

6.2. Dominio *Doughnut*

Este conjunto de datos, conocido con el nombre de *doughnut,* ha sido generado de forma artificial, es decir, los datos no proceden del mundo real. Este dominio contiene 10.000 instancias. Inicialmente este conjunto de datos ha sido creado con dos atributos y uno adicional que contiene la clase de cada patrón. Cabe destacar que existen dos clases: TRUE y FALSE. Como se puede apreciar en la *ilustración 6.1*, su nombre es debido a que los datos que pertenecen a la clase TRUE describen una forma similar a un *doughnut*.

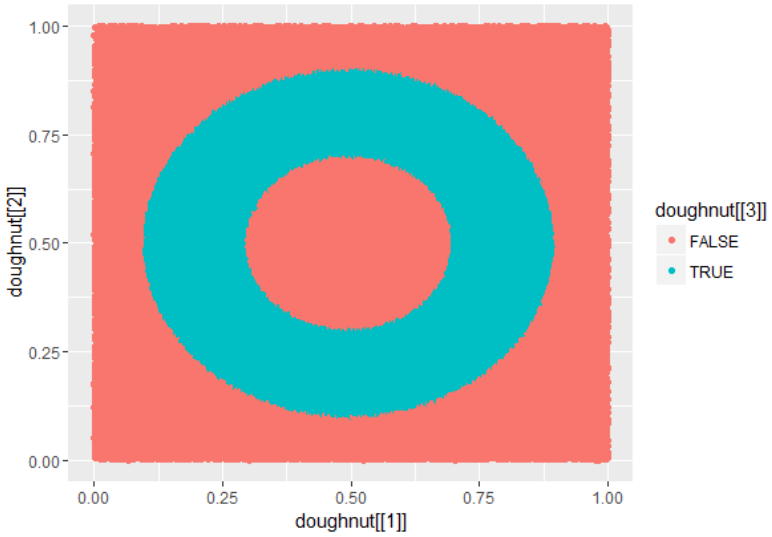

Ilustración 6.1: *Conjunto de datos llamado doughnut*

6.2.1. Experimentos realizados sobre el dominio *Doughnut*

6.2.1.1. Experimentos añadiendo 8 atributos adicionales

En las primeras pruebas realizadas en este dominio serán añadidos ocho atributos que contendrán valores aleatorios en el intervalo [0,1]. Por tanto, el nuevo conjunto de datos tendrá un total de 10 atributos y la clase de salida, ya que tanto los dos atributos iniciales como la clase se mantendrán con los valores iniciales.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.6731
3	0.6980
5	0.7078
7	0.7144

Tabla 6.1: *Resultados de KNN sobre los datos originales en Doughnut 8*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.2* se muestra el resultado que proporciona KNN sobre la reducción de la dimensionalidad que produce la transformación lineal de la capa oculta de la red en el conjunto de datos llamado *Doughnut*, habiendo añadido 8 atributos con valores aleatorios.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
1	0.5310	0.5449	0.5535	0.5611	0.5890	0.6023	0.6108	0.6176
2	0.8304	0.8446	0.8500	0.8536	0.8596	0.8646	0.8671	0.8694
3	0.9842	0.9856	0.9858	0.9863	0.9913	0.9913	0.9916	0.9916
4	0.9769	0.9795	0.9807	0.9807	0.9914	0.9923	0.9922	0.9922
5	0.9836	0.9851	0.9856	0.9858	0.9925	0.9929	0.9931	0.9929
6	0.9814	0.9832	0.9838	0.9844	0.9890	0.9888	0.9887	0.9885
7	0.9737	0.9763	0.9778	0.9791	0.9703	0.9715	0.9717	0.9719

Tabla 6.2: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Doughnut 8*

En la *Tabla 6.2* se muestra que el mejor resultado logrado con la transformación lineal y la red de neuronas entrenada con 100 iteraciones es de 98,63%, mientras que el mejor resultado de la red entrenada con 1.000 iteraciones corresponde con 99,31%. Por tanto, el mejor resultado que se obtiene con la reducción de la dimensionalidad realizada por la transformación lineal es de 99,31% con 5 neuronas en la capa oculta y 5 vecinos.

En la *Tabla 6.3* se pueden observar los resultados obtenidos por KNN sobre la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
1	0.5286	0.5443	0.5538	0.5619	0.5889	0.6025	0.6108	0.6174
2	0.8208	0.8291	0.8344	0.8377	0.8550	0.8610	0.8632	0.8654
3	0.9826	0.9841	0.9847	0.9849	0.9890	0.9892	0.9896	0.9896
4	0.9769	0.9793	0.9798	0.9802	0.9873	0.9881	0.9878	0.9877
5	0.9851	0.9864	0.9871	0.9873	0.9902	0.9905	0.9906	0.9907
6	0.9802	0.9818	0.9823	0.9830	0.9904	0.9906	0.9905	0.9904
7	0.9738	0.9762	0.9772	0.9785	0.9873	0.9875	0.9876	0.9877

Tabla 6.3: *Resultados de KNN sobre la transformación lineal normalizada en Doughnut 8*

Comparando los valores máximos obtenidos con 100 y 1.000 iteraciones de la tabla anterior se puede concluir que la tasa de acierto más elevada, en el caso de la transformación lineal normalizada, es de 99,07%. Este valor es registrado con 5 neuronas y 7 vecinos.

En la siguiente tabla se muestra la tasa media de acierto ofrecida por KNN sobre los datos del conjunto de test con la reducción de la dimensionalidad aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
1	0.4445	0.4517	0.4544	0.4580	0.5057	0.5040	0.5090	0.5168
2	0.7614	0.7697	0.7739	0.7764	0.7389	0.7417	0.7423	0.7426
3	0.8736	0.8703	0.8686	0.8673	0.8647	0.8586	0.8548	0.8512
4	0.8740	0.8807	0.8822	0.8827	0.9240	0.9199	0.9171	0.9150
5	0.9204	0.9214	0.9207	0.9193	0.9664	0.9651	0.9638	0.9621
6	0.9298	0.9313	0.9301	0.9293	0.9863	0.9846	0.9833	0.9822
7	0.9455	0.9437	0.9416	0.9394	0.9950	0.9948	0.9946	0.9942

Tabla 6.4: *Resultados de KNN aplicando la función sigmoidal en Doughnut 8*

En la *Tabla 6.4* se puede comprobar que el porcentaje máximo de acierto que se obtiene con la red de neuronas entrenada con 100 iteraciones es de 94,55%, habiendo aplicado la función sigmoidal para realizar la reducción de la dimensionalidad. Sin embargo, con la red entrenada con 1.000 iteraciones se consigue un mejor resultado, alcanzando un 99,50% de acierto con 7 neuronas en la capa oculta.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Vecinos / Componentes	1	3	5	7
1	0.5284	0.5435	0.5530	0.5600
2	0.5321	0.5468	0.5556	0.5638
3	0.5381	0.5503	0.5603	0.5678
4	0.5348	0.5526	0.5617	0.5685
5	0.5489	0.5604	0.5686	0.5737
6	0.5940	0.6128	0.6235	0.6313
7	0.5883	0.6110	0.6225	0.6288

Tabla 6.5: *Resultados de KNN sobre los datos transformados por PCA Doughnut 8*

6.2.1.2. Experimentos añadiendo 8 atributos adicionales y rotando los datos

En este apartado se presentan los resultados obtenidos aplicando una rotación sobre los datos del apartado anterior, generando un nuevo conjunto de datos con nuevamente 10 atributos y la clase. Analizar esta situación es muy interesante puesto que se añade más aleatoriedad al problema y se podrá comprobar si la red de neuronas es capaz de extraer la información importante.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.6731
3	0.6980
5	0.7078
7	0.7144

Tabla 6.6: *Resultados de KNN sobre los datos originales en Doughnut Rotados 8*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.7* se puede apreciar la tasa media de acierto que proporciona KNN sobre la transformación lineal que realiza la capa oculta de la red en el conjunto de datos *Doughnut Rotados 8*.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
1	0.5612	0.5739	0.5823	0.5885	0.5293	0.5422	0.5521	0.5614
2	0.8278	0.8368	0.8401	0.8410	0.8507	0.8562	0.8581	0.8598
3	0.9873	0.9881	0.9885	0.9888	0.9919	0.9922	0.9923	0.9923
4	0.9807	0.9823	0.9832	0.9833	0.9901	0.9911	0.9925	0.9907
5	0.9763	0.9787	0.9795	0.9802	0.9918	0.9924	0.9745	0.9925
6	0.9770	0.9793	0.9805	0.9832	0.9939	0.9939	0.9937	0.9854
7	0.9822	0.9842	0.9850	0.9858	0.9717	0.9736	0.9745	0.9749

Tabla 6.7: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Doughnut Rotados 8*

En la tabla se observa que en la reducción de la dimensionalidad producida con la transformación lineal, habiendo realizado el entrenamiento con 100 iteraciones, el mejor resultado registrado es de 98,88%, mientras que entrenando la red con 1.000 iteraciones el resultado más elevado es de 99,39%. Por tanto, en este caso el porcentaje más alto de acierto obtenido es de 99,39% con 6 neuronas en la capa oculta.

En la siguiente tabla se muestra el resultado de aplicar KNN sobre la reducción de la dimensionalidad producida por la transformación lineal normalizada. En este caso los resultados mostrados son sobre el dominio llamado *Doughnut* al que se le han añadido 8 atributos con valores aleatorios y se ha realizado una rotación de los datos.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
1	0.5615	0.5746	0.5827	0.5888	0.5302	0.5439	0.5536	0.5611
2	0.8264	0.8342	0.8370	0.8390	0.8489	0.8540	0.8566	0.8580
3	0.9863	0.9874	0.9877	0.9878	0.9901	0.9906	0.9905	0.9907
4	0.9796	0.9813	0.9821	0.9824	0.9883	0.9892	0.9889	0.9889
5	0.9788	0.9806	0.9818	0.9823	0.9894	0.9899	0.9896	0.9899
6	0.9759	0.9789	0.9798	0.9805	0.9909	0.9907	0.9904	0.9861
7	0.9748	0.9774	0.9818	0.9802	0.9867	0.9872	0.9876	0.9881

Tabla 6.8: *Resultados de KNN sobre la transformación lineal normalizada en Doughnut Rotados 8*

En la *Tabla 6.8* se pueden comparar los valores obtenidos con la transformación lineal normalizada, ejecutando el entrenamiento de la red de neuronas con 100 iteraciones y 1.000 iteraciones. Se puede apreciar que con 1.000 iteraciones se logra un mejor resultado en la tasa de acierto, concretamente se alcanza un 99,07% con 3 neuronas en la capa oculta.

En la *Tabla 6.9* se muestran los resultados proporcionados por KNN sobre la reducción de la dimensionalidad realizada por la red de neuronas, aplicando la función sigmoidal, en *Doughnut Rotados 8*.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
1	0.5372	0.5482	0.5553	0.5618	0.5293	0.5422	0.5521	0.5614
2	0.7638	0.7665	0.7680	0.7685	0.7528	0.7316	0.7342	0.7351
3	0.8458	0.8473	0.8485	0.8478	0.8635	0.8564	0.8513	0.8482
4	0.8548	0.8570	0.8562	0.8567	0.9470	0.9425	0.9399	0.9370
5	0.9165	0.9172	0.9163	0.9145	0.9579	0.9548	0.9523	0.9499
6	0.9147	0.9172	0.9163	0.9142	0.9638	0.9599	0.9574	0.9424
7	0.9501	0.9501	0.9494	0.9486	0.9835	0.9817	0.9808	0.9802

Tabla 6.9: *Resultados de KNN aplicando la función sigmoidal en Doughnut Rotados 8*

Por último, podemos observar en la *Tabla 6.9* que aplicando la función sigmoidal y entrenando la red con 100 iteraciones el mejor resultado obtenido es de 95.01%. Sin embargo, realizando el entrenamiento con 1.000 iteraciones se mejora el valor anterior, consiguiendo un 98,35%. Teniendo en cuenta estos datos, el mejor resultado aplicando la función sigmoidal es 98,35% que se registra con 7 neuronas en la capa oculta.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Vecinos / Componentes	1	3	5	7
1	0.5284	0.5435	0.5530	0.5600
2	0.5321	0.5468	0.5556	0.5638
3	0.5381	0.5503	0.5603	0.5678
4	0.5348	0.5526	0.5617	0.5685
5	0.5489	0.5604	0.5686	0.5737
6	0.5940	0.6128	0.6235	0.6313
7	0.5884	0.6110	0.6225	0.6288

Tabla 6.10: *Resultados de KNN sobre los datos transformados por PCA en Doughnut Rotados 8*

6.2.2. Análisis de los resultados obtenidos en el dominio *Doughnut*

6.2.2.1. Análisis de los resultados obtenidos en los experimentos añadiendo 8 atributos adicionales

En la *Tabla 6.1* se puede observar que la tasa de acierto de clasificación obtenida sobre los datos originales es de 71,44%.

Como se comentó anteriormente, la tasa de acierto más alta producida por la transformación lineal es de 99,31% con 5 neuronas en la capa oculta. Si comparamos este porcentaje de acierto con el conseguido sobre los datos originales (71,44%), podemos observar que se obtiene un mejor resultado en la clasificación realizando la reducción de la dimensionalidad con la transformación lineal. Además, cabe destacar que la mejora se produce con un conjunto de datos con 5 atributos frente a los 10 atributos que contenía el conjunto de datos original.

Un resultado similar al caso anterior es obtenido por la transformación lineal normalizada, ya que se alcanza un 99,07% con 5 neuronas en la capa oculta. Por tanto, en este caso también se mejoran los resultados de clasificación obtenidos sobre el conjunto de datos original (71,44%).

Con la reducción de la dimensionalidad producida por la red de neuronas, aplicando la función sigmoidal, el mejor resultado alcanzado es de 99,5% (véase *Tabla 6.4*) con 7 neuronas. Con este método incluso se logran unos mejores resultados que con la transformación lineal y la transformación lineal normalizada y, por tanto, también se mejora el resultado de clasificación sobre los datos originales (la mejora que se produce es de 28,06%).

Si observamos la *Tabla 6.5* vemos que el mejor resultado que se obtiene con PCA es de 63,13%. Como se puede apreciar, este valor no mejora el resultado alcanzado sobre los datos originales (71,44%). Además, también se observa que si se realizara la reducción de la dimensionalidad con PCA, se perdería un 36,37% en la tasa de acierto con respecto a la producida con la red de neuronas (aplicando la función sigmoidal).

A continuación, se va mostrar una comparación más detallada de todos los resultados proporcionados por KNN sobre la reducción de la dimensionalidad llevada a cabo por PCA y la red de neuronas. Para realizar esto, se calculará la resta de los resultados obtenidos por la red de neuronas, entrenada con 100 y 1.000 iteraciones, y los resultados obtenidos por PCA. Por tanto, la mejora será mayor cuanto mayor sean los valores obtenidos en la resta. Además, cabe destacar que esta comparación será llevada a cabo por separado con las tres versiones de reducción de la dimensionalidad hechas por la red de neuronas (transformación lineal, transformación lineal normalizada y aplicando la función sigmoidal).

Comparación entre los resultados de la transformación lineal y PCA

En la *Tabla 6.11* se muestra la diferencia de la tasa de acierto obtenida por la transformación lineal de la red de neuronas y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
1	0.0026	0.0014	0.0005	0.0011	0.0606	0.0588	0.0578	0.0576
2	0.2983	0.2978	0.2944	0.2898	0.3275	0.3178	0.3115	0.3056
3	0.4461	0.4353	0.4255	0.4185	0.4532	0.4410	0.4313	0.4238
4	0.4421	0.4269	0.4190	0.4122	0.4566	0.4397	0.4305	0.4237
5	0.4347	0.4247	0.4170	0.4121	0.4436	0.4325	0.4245	0.4192
6	0.3874	0.3704	0.3603	0.3531	0.3950	0.3760	0.3652	0.3572
7	0.3854	0.3653	0.3553	0.3503	0.3820	0.3605	0.3492	0.3431

Tabla 6.11: *Comparación entre los resultados de la transformación lineal y PCA en Doughnut 8*

En la tabla comparativa se puede apreciar claramente que los resultados de la reducción de la dimensionalidad producidos por la transformación lineal son muy superiores a los que se obtienen con PCA y, además, se mejoran todos los casos. Entrenando la red de neuronas con 1.000 iteraciones el porcentaje medio de mejora que se alcanza es del 34,45% y el valor máximo obtenido es 45,66%. Sin embargo, con la red de neuronas entrenada con 100 iteraciones, aunque se siguen superando los resultados de PCA de forma amplia, se consigue una mejora ligeramente menor que la anterior, ya que la media obtenida es de 32,96% y el máximo es de 44,61%.

En la siguiente gráfica se muestra la evolución de ambas aproximaciones en el dominio *Doughnut 8*. En la gráfica se observa que los resultados proporcionados por la transformación lineal son claramente superiores a los que se obtienen con PCA.

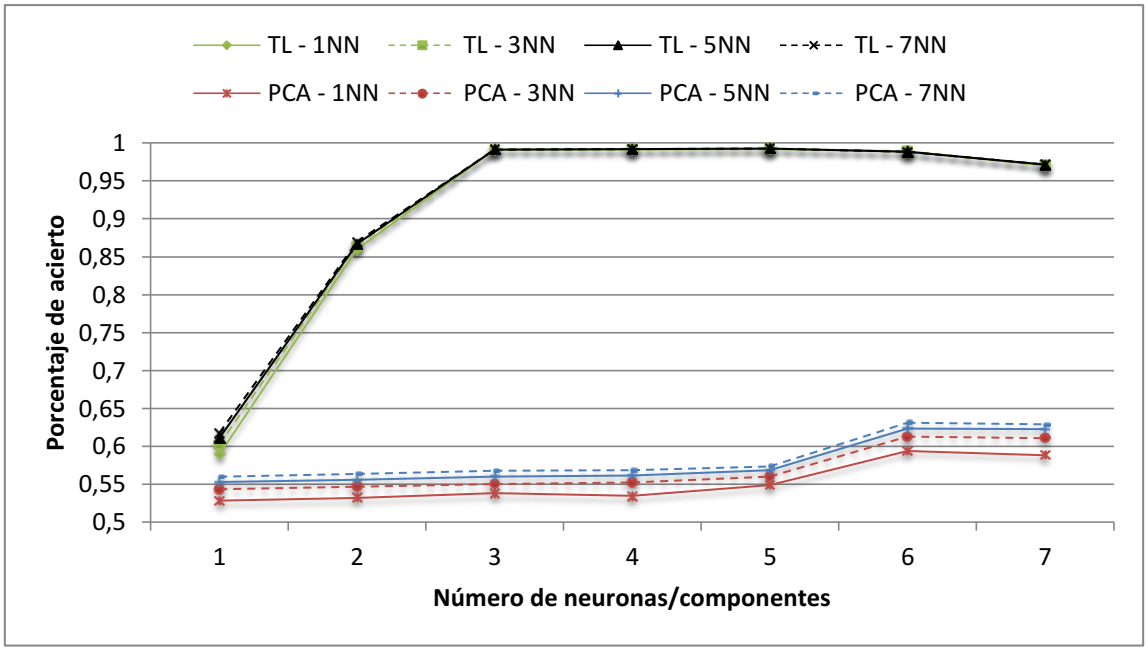

Ilustración 6.2: *Gráfica comparativa de la transformación lineal y PCA en Doughnut 8*

Comparación entre los resultados de la transformación lineal normalizada y PCA

En la siguiente tabla se muestra la comparación realizada entre la transformación lineal normalizada y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
1	0.0026	0.0014	0.0005	0.0011	0.0605	0.0590	0.0578	0.0574
2	0.2983	0.2978	0.2944	0.2898	0.3229	0.3142	0.3076	0.3016
3	0.4461	0.4353	0.4255	0.4185	0.4509	0.4389	0.4293	0.4218
4	0.4421	0.4269	0.4190	0.4122	0.4525	0.4355	0.4261	0.4192
5	0.4347	0.4247	0.4170	0.4121	0.4413	0.4301	0.4220	0.4170
6	0.3874	0.3704	0.3603	0.3531	0.3964	0.3778	0.3670	0.3591
7	0.3854	0.3653	0.3553	0.3503	0.3990	0.3765	0.3651	0.3589

Tabla 6.12: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Doughnut 8*

Nuevamente, en la *Tabla 6.12* se puede comprobar que se mejoran los resultados conseguidos por PCA en todos los casos. Con una neurona en la capa oculta se observa que la mejora no es muy significativa, pero ya con 2 neuronas los resultados son mucho más amplios.

Por tanto, el porcentaje medio de mejora que se obtiene con la red entrenada con 1.000 iteraciones es de 34,52% y el valor máximo de mejora alcanzado es de 45,25%. Con la red entrenada con 100 iteraciones se logra un 32,96% de mejora media y un máximo de 44,61%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la siguiente tabla podemos observar una tabla comparativa de la reducción de la dimensionalidad hecha con la red de neuronas, aplicando la función sigmoidal, y PCA. La tabla ha sido obtenida calculando la resta entre ambas aproximaciones, por tanto, valores altos en la tabla significará una mayor mejora.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
1	-0.0839	-0.0918	-0.0986	-0.1020	-0.0227	-0.0395	-0.0440	-0.0432
2	0.2293	0.2229	0.2183	0.2126	0.2068	0.1949	0.1867	0.1788
3	0.3355	0.3200	0.3083	0.2995	0.3266	0.3083	0.2945	0.2834
4	0.3392	0.3281	0.3205	0.3142	0.3892	0.3673	0.3554	0.3465
5	0.3715	0.3610	0.3521	0.3456	0.4175	0.4047	0.3952	0.3884
6	0.3358	0.3185	0.3066	0.2980	0.3923	0.3718	0.3598	0.3509
7	0.3572	0.3327	0.3191	0.3106	0.4067	0.3838	0.3721	0.3654

Tabla 6.13: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Doughnut 8*

En este caso, los resultados que se obtienen son ligeramente peores con 1 neurona pero a medida que aumentamos el número de neuronas se consiguen mejores resultados, llegando a lograr un valor máximo de mejora de 41,75% con la red entrenada con 1.000 iteraciones. Por otro lado, el valor máximo de mejora alcanzado con 100 iteraciones es de 37,15% que también es muy elevado.

Por tanto, se obtienen peores resultados en 8 situaciones y, por el contrario, se mejoran los resultados en 48 situaciones. A pesar de esto, cabe destacar que el porcentaje medio de mejora con respecto a PCA es 28,21% con la red entrenada con 1.000 iteraciones y un 25,29% con la red entrenada con 100 iteraciones.

6.2.2.2. Análisis de los resultados obtenidos en los experimentos añadiendo 8 atributos adicionales y rotando los datos

Como se observa en la *Tabla 6.6*, la tasa de acierto ofrecida por KNN sobre este conjunto de datos con 10 atributos es de 71,44%.

En la reducción de la dimensionalidad producida por la transformación lineal se obtiene un resultado máximo de 99,39% con 6 neuronas en la capa oculta. Si comparamos este resultado con el porcentaje registrado sobre el conjunto de datos original (71,44%), se puede apreciar que se produce una amplia mejora, concretamente de un 27,93%.

Con la transformación lineal normalizada también se obtiene mejora con respecto a la clasificación llevada a cabo sobre los datos originales (71,44%), ya que se alcanza un valor máximo de 99,07% con 3 neuronas en la capa oculta. Cabe destacar que se produce un resultado ligeramente inferior comparándolo con el resultado obtenido con la transformación lineal (99,39%), pero este valor es registrado con 6 neuronas mientras que en el caso de la transformación lineal normalizada el valor máximo es registrado con 3 neuronas, es decir, con la transformación lineal normalizada el nuevo conjunto de datos tendría únicamente 3 atributos.

Por último, con la reducción de la dimensionalidad producida aplicando la función sigmoidal se alcanza una tasa de acierto máxima de un 98,35% con 7 neuronas en la capa oculta. Como se puede observar, este valor mejora el porcentaje de acierto obtenido por KNN sobre los datos originales (71,44%). Por otro lado, cabe destacar que en este dominio la función sigmoidal no ofrece unos resultados tan buenos como la transformación lineal y la transformación lineal normalizada, ya que no solo se produce un porcentaje de éxito menor sino que el conjunto de datos transformados con un número de dimensiones reducidas tendría 7 atributos.

Adicionalmente, se observa en la *Tabla 6.10* que el mejor resultado que proporciona PCA es de 63,13% con 6 componentes. Nótese que se empeoran los resultados de clasificación de los datos originales (71,44%) y los resultados obtenidos realizando la reducción de la dimensionalidad con la red de neuronas (99,39%, 99,07% y 98,35%).

A continuación, se muestra una comparativa más amplia entre los resultados obtenidos por la reducción de la dimensionalidad realizada por la transformación lineal, la transformación lineal normalizada y aplicando la función sigmoidal con PCA. La comparación es llevada a cabo por separado con la red entrenada con 100 y 1.000 iteraciones.

Comparación entre los resultados de la transformación lineal y PCA

En la *Tabla 6.14* se muestra la comparación calculada de la transformación lineal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
1	0.0328	0.0304	0.0293	0.0285	0.0009	-0.0013	-0.0009	0.0014
2	0.2957	0.2900	0.2845	0.2772	0.3186	0.3094	0.3025	0.2960
3	0.4492	0.4378	0.4282	0.4210	0.4538	0.4419	0.4320	0.4245
4	0.4459	0.4297	0.4215	0.4148	0.4553	0.4385	0.4308	0.4222
5	0.4274	0.4183	0.4109	0.4065	0.4429	0.4320	0.4059	0.4188
6	0.3830	0.3665	0.3570	0.3519	0.3999	0.3811	0.3702	0.3541
7	0.3938	0.3732	0.3625	0.3570	0.3833	0.3626	0.3520	0.3461

Tabla 6.14: *Comparación entre los resultados de la transformación lineal y PCA en Doughnut Rotados 8*

En la anterior tabla se observa que con la red entrenada con 1.000 iteraciones existen dos situaciones en las que los resultados de PCA son mejores que los de la transformación lineal, sin embargo la pérdida que se produce en ambos casos es ínfima (0,13% y 0,09%). En el resto de los casos se mejoran los resultados de una forma muy amplia, llegando a conseguir una mejora máxima del 45,53%. Adicionalmente, la tasa media de acierto que se mejora es de 33,48%.

En el caso de la red entrenada con 100 iteraciones se observa que se mejoran todos los resultados de PCA. La mejora media obtenida en este caso es de 33,30% y el valor máximo de mejora es de 44,92%.

A continuación, se muestra la gráfica comparativa de la transformación lineal y PCA en el dominio *Doughnut rotados 8*. Nuevamente, en la gráfica se observa la gran diferencia existente entre ambas aproximaciones, logrando con la transformación lineal unos resultados muy superiores.

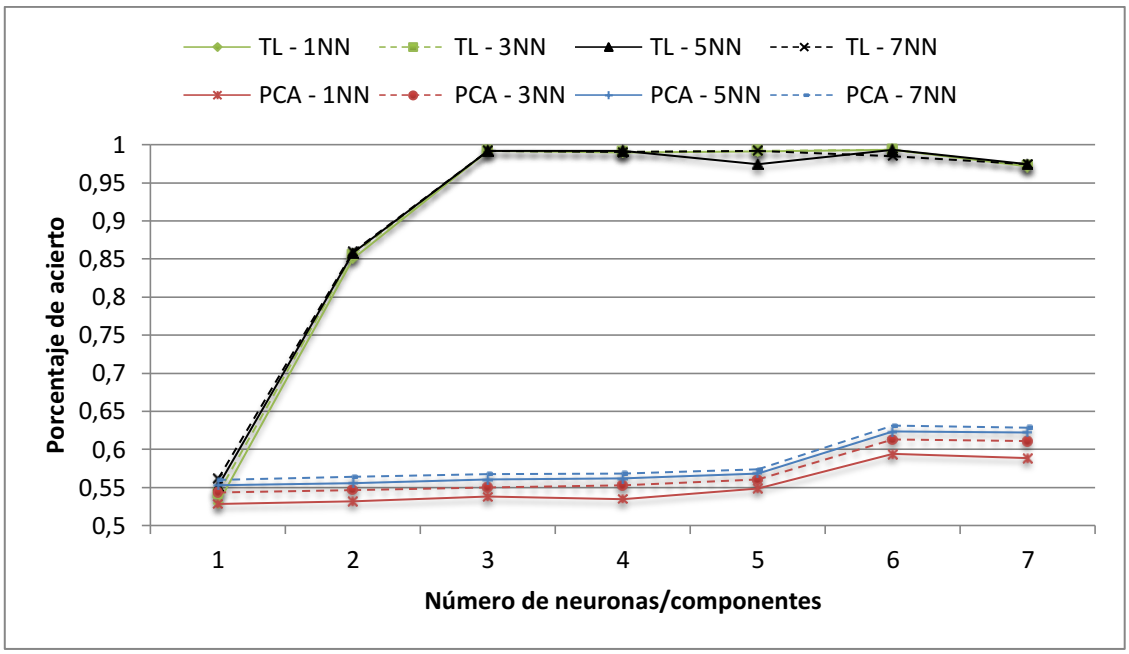

Ilustración 6.3: *Gráfica comparativa de la transformación lineal y PCA en Doughnut rotados 8*

Comparación entre los resultados de la transformación lineal normalizada y PCA

En la *Tabla 6.15* se muestra la tabla comparativa de la reducción de la dimensionalidad producida con la transformación lineal normalizada y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
1	0.0331	0.0311	0.0297	0.0288	0.0018	0.0004	0.0006	0.0011
2	0.2943	0.2874	0.2814	0.2752	0.3168	0.3072	0.3010	0.2942
3	0.4482	0.4371	0.4274	0.4200	0.4520	0.4403	0.4302	0.4229
4	0.4448	0.4287	0.4204	0.4139	0.4535	0.4366	0.4272	0.4204
5	0.4299	0.4202	0.4132	0.4086	0.4405	0.4295	0.4210	0.4162
6	0.3819	0.3661	0.3563	0.3492	0.3969	0.3779	0.3669	0.3548
7	0.3864	0.3664	0.3593	0.3514	0.3983	0.3762	0.3651	0.3593

Tabla 6.15: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Doughnut Rotados 8*

Los resultados que se obtienen en la comparación son todos positivos, por lo que en todas las situaciones se mejoran los resultados de PCA. Con la red entrenada con 1.000 iteraciones el porcentaje medio de mejora es de 33,60% y el valor máximo alcanzado es de 45,35%, mientras que con la red entrenada con 100 iteraciones la mejora media producida es de 33,18% y el valor máximo de mejora corresponde con 44,82%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la siguiente tabla se muestran los resultados de aplicar la resta de la tasa de acierto obtenida en la reducción de la dimensionalidad realizada aplicando la función sigmoidal y la tasa de acierto registrada con PCA. Es importante destacar que la mejora producida será mayor cuanto mayor sean los valores de la tabla y, por el contrario, si se obtienen resultados negativos, la solución propuesta empeorará los resultados de PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
1	0.0088	0.0047	0.0023	0.0018	0.0009	-0.0013	-0.0009	0.0014
2	0.2317	0.2197	0.2124	0.2047	0.2207	0.1848	0.1786	0.1713
3	0.3077	0.2970	0.2882	0.2800	0.3254	0.3061	0.2910	0.2804
4	0.3200	0.3044	0.2945	0.2882	0.4122	0.3899	0.3782	0.3685
5	0.3676	0.3568	0.3477	0.3408	0.4090	0.3944	0.3837	0.3762
6	0.3207	0.3044	0.2928	0.2829	0.3698	0.3471	0.3339	0.3111
7	0.3617	0.3391	0.3269	0.3198	0.3951	0.3707	0.3583	0.3514

Tabla 6.16: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Doughnut Rotados 8*

En la tabla se puede observar que se encuentran 2 casos en los que los resultados se empeoran ligeramente. Por el contrario, en los restantes 54 casos los resultados son superiores y en algunas ocasiones la mejora que se produce es muy significativa, alcanzando un máximo de 41,22% con la red entrenada con 1.000 iteraciones y 36,76% con la red entrenada con 100 iteraciones. Por otro lado, merece la pena destacar que la mejora media que se produce es de 28,24% y 25,81%, respectivamente.

6.3. Dominio *Spambase*

Este dominio, donado por George Forman y conocido con el nombre de *Spambase*, contiene una base de datos de correos electrónicos que han sido etiquetados como correos deseados o no deseados (spam). En total han sido registrados un total de 4.601 correos electrónicos.

El conjunto de datos está formado por 57 atributos que contienen fundamentalmente si una palabra o carácter es encontrado de forma frecuente en el correo electrónico. A continuación, se explican los atributos que contiene el dominio:

- Los primeros 48 atributos son atributos de tipo numérico real en el intervalo [0,100] y corresponden con el porcentaje de aparición de una palabra concreta en el correo electrónico. Una palabra en este caso es una cadena de caracteres alfanuméricos delimitados por caracteres no alfanuméricos o de fin de cadena.
- Los siguientes 6 atributos son atributos de tipo numérico real en el intervalo [0,100] y contienen el porcentaje de aparición de caracteres en el correo electrónico.
- Contiene un atributo de tipo entero numérico en el intervalo [1,∞) y corresponde con el tamaño de la secuencia ininterrumpida más grande de letras mayúsculas que hay en el correo electrónico.
- El último atributo es de tipo entero numérico en el intervalo [1,∞) y corresponde con la suma de todas las secuencias ininterrumpidas de letras mayúsculas.

6.3.1. Experimentos realizados sobre el dominio *Spambase*

6.3.1.1. Experimentos sobre los datos originales

En este apartado se muestran los resultados obtenidos de los experimentos ejecutados sobre el conjunto de datos *Spambase* original, es decir, en este caso se tienen 57 dimensiones y la clase de cada patrón.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.8929
3	0.8951
5	0.8957
7	0.8924

Tabla 6.17: *Resultados de KNN sobre los datos originales en Spambase*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.18* se muestran los resultados obtenidos por KNN sobre la transformación lineal que producen las neuronas de la capa oculta de la red.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
4	0.9138	0.9209	0.9250	0.9243	0.9160	0.9212	0.9237	0.9251
8	0.9204	0.9252	0.9274	0.9276	0.9202	0.9264	0.9264	0.9273
12	0.9151	0.9189	0.9213	0.9215	0.9163	0.9225	0.9263	0.9292
16	0.9165	0.9238	0.9247	0.9239	0.9173	0.9254	0.9246	0.9252
20	0.9170	0.9201	0.9218	0.9221	0.9162	0.9200	0.9244	0.9243
24	0.9202	0.9229	0.9246	0.9262	0.9236	0.9241	0.9272	0.9268
28	0.9216	0.9243	0.9255	0.9270	0.9138	0.9198	0.9237	0.9237

Tabla 6.18: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Spambase*

Como se puede observar en la *Tabla 6.18,* el mejor resultado que se registra con la transformación lineal y la red entrenada con 100 iteraciones es de 92,76%, mientras que con 1.000 iteraciones el mejor resultado obtenido es 92,92%. Por tanto, el mejor resultado de reducción de dimensionalidad con la transformación lineal es 92,92% que se ofrece con 12 neuronas en la capa oculta y 7 vecinos.

En la *Tabla 6.19* se muestran los resultados producidos por KNN, habiendo normalizado en el intervalo [0,1] la transformación lineal que realizan las neuronas de la capa oculta de la red.

Neuronas \ Vecinos	100 iteraciones				1.000 iteraciones			
	1	3	5	7	1	3	5	7
4	0.6993	0.7130	0.7220	0.7250	0.6508	0.6774	0.6915	0.7064
8	0.7761	0.7926	0.7995	0.8029	0.7549	0.7479	0.7591	0.7625
12	0.8264	0.8387	0.8441	0.8450	0.7803	0.8063	0.8332	0.8382
16	0.7907	0.8212	0.8218	0.8290	0.7974	0.8090	0.8173	0.8213
20	0.8112	0.8250	0.8305	0.8339	0.8370	0.8465	0.8556	0.8608
24	0.8237	0.8396	0.8435	0.8461	0.8064	0.8171	0.8351	0.8370
28	0.8566	0.8635	0.8787	0.8826	0.8027	0.8178	0.8264	0.8336

Tabla 6.19: *Resultados de KNN sobre la transformación lineal normalizada en Spambase*

En la tabla anterior se observa que el mejor resultado que se alcanza con la red entrenada con 100 iteraciones es 88,26%. Sin embargo, con la red entrenada con 1.000 iteraciones se mejora el resultado anterior. Por ello, el máximo valor con la transformación lineal normalizada es 88,26% y se obtiene con 28 neuronas en la capa oculta y 7 vecinos.

En la *Tabla 6.20* se muestran los resultados obtenidos por KNN sobre la reducción de la dimensionalidad producida aplicando la función sigmoidal.

Neuronas \ Vecinos	100 iteraciones				1.000 iteraciones			
	1	3	5	7	1	3	5	7
4	0.9180	0.9264	0.9295	0.9298	0.9171	0.9273	0.9292	0.9304
8	0.9187	0.9273	0.9304	0.9320	0.9225	0.9274	0.9274	0.9278
12	0.9172	0.9216	0.9248	0.9241	0.9174	0.9236	0.9242	0.9267
16	0.9209	0.9245	0.9263	0.9266	0.9207	0.9267	0.9283	0.9288
20	0.9171	0.9208	0.9257	0.9262	0.9162	0.9190	0.9198	0.9223
24	0.9178	0.9247	0.9255	0.9261	0.9201	0.9199	0.9215	0.9209
28	0.9186	0.9215	0.9237	0.9257	0.9158	0.9190	0.9219	0.9220

Tabla 6.20: *Resultados de KNN aplicando la función sigmoidal en Spambase*

Por último, podemos observar en la *Tabla 6.20* que aplicando la función sigmoidal y entrenando la red con 100 iteraciones el mejor resultado obtenido es de 93,20%, y realizando el entrenamiento con 1.000 iteraciones no se mejora el valor máximo anterior, consiguiendo únicamente un 93,04%. Por tanto, el mejor resultado aplicando la función sigmoidal es 93,20% y se registra con 8 neuronas en la capa oculta y 7 vecinos.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
4	0.8679	0.8712	0.8826	0.8891
8	0.9005	0.8989	0.9103	0.9082
12	0.9016	0.9120	0.9098	0.9152
16	0.9060	0.9120	0.9163	0.9185
20	0.9120	0.9158	0.9103	0.9136
24	0.9103	0.9130	0.9087	0.9109
28	0.9043	0.9103	0.9098	0.9120

Tabla 6.21: *Resultados de KNN sobre los datos transformados por PCA en Spambase*

6.3.1.2. Experimentos añadiendo 8 atributos adicionales y rotando los datos

En este apartado se muestran los resultados producidos añadiendo, al conjunto original de datos, ocho atributos adicionales con valores aleatorios en el intervalo [0,1]. Posteriormente, se ha hecho una rotación de los datos. Por tanto, se tiene un conjunto de datos con 65 atributos (57 atributos originales y 8 atributos aleatorios) y la clase para cada patrón.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.6609
3	0.6679
5	0.6766
7	0.6842

Tabla 6.22: *Resultados de KNN sobre los datos originales en Spambase Rotados 8*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.23* se pueden apreciar los resultados arrojados por KNN aplicado sobre la transformación lineal que lleva a cabo las neuronas de la capa oculta de la red.

	100 iteraciones				1.000 iteraciones			
Neuronas \ Vecinos	1	3	5	7	1	3	5	7
2	0.8945	0.9185	0.9215	0.9221	0.8948	0.9152	0.9187	0.9201
8	0.8777	0.9002	0.9037	0.9095	0.8709	0.8920	0.9008	0.9040
14	0.8628	0.8906	0.8967	0.8996	0.8593	0.8879	0.8920	0.8952
20	0.8404	0.8708	0.8784	0.8846	0.8500	0.8816	0.8910	0.8971
26	0.8407	0.8716	0.8803	0.8864	0.8442	0.8734	0.8820	0.8859
32	0.8347	0.8670	0.8775	0.8815	0.8380	0.8652	0.8772	0.8818
38	0.8322	0.8608	0.8726	0.8782	0.8309	0.8627	0.8749	0.8821

Tabla 6.23: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Spambase Rotados 8*

En la *Tabla 6.23* se puede observar que el valor máximo obtenido con la red entrenada con 100 iteraciones es 92,21%, mientras que con 1.000 iteraciones el mejor resultado es 92,01%. Por ello, en este caso el mejor resultado conseguido con la transformación lineal es 92,21% con 2 neuronas y 7 vecinos.

En la siguiente tabla se muestran los resultados obtenidos por KNN sobre la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.7286	0.7380	0.7421	0.7397	0.6742	0.6822	0.6818	0.6907
8	0.7616	0.7908	0.8050	0.8111	0.7620	0.7877	0.7991	0.8021
14	0.7660	0.7985	0.8104	0.8177	0.7424	0.7714	0.7772	0.7842
20	0.7924	0.8260	0.8335	0.8359	0.7747	0.8014	0.8136	0.8230
26	0.7712	0.7951	0.8041	0.8102	0.7789	0.8076	0.8204	0.8236
32	0.7792	0.8029	0.8134	0.8147	0.7740	0.8023	0.8105	0.8149
38	0.7841	0.8130	0.8236	0.8309	0.7849	0.8228	0.8351	0.8408

Tabla 6.24: *Resultados de KNN sobre la transformación lineal normalizada en Spambase Rotados 8*

El mejor resultado registrado con la transformación lineal normalizada y la red entrenada con 100 iteraciones es 83,59%. Sin embargo, entrenando la red con 1.000 iteraciones se alcanza un 84,08% que es superior al valor anterior. Por tanto, el mejor porcentaje de acierto logrado con la transformación lineal normalizada es 84,08% que corresponde con 38 neuronas en la capa oculta y 7 vecinos.

En la siguiente tabla se pueden observar los resultados arrojados por KNN sobre la reducción de dimensionalidad producida aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.8933	0.9176	0.9208	0.9213	0.8941	0.9167	0.9195	0.9191
8	0.8910	0.9126	0.9164	0.9170	0.8894	0.9106	0.9131	0.9127
14	0.8911	0.9090	0.9108	0.9099	0.8823	0.8983	0.9013	0.9024
20	0.8837	0.8975	0.8998	0.9015	0.8721	0.8875	0.8912	0.8934
26	0.8830	0.9025	0.9064	0.9053	0.8843	0.8979	0.9001	0.9002
32	0.8718	0.8861	0.8921	0.8932	0.8720	0.8873	0.8923	0.8939
38	0.8566	0.8778	0.8815	0.8843	0.8660	0.8858	0.8901	0.8948

Tabla 6.25: *Resultados de KNN aplicando la función sigmoidal en Spambase Rotados 8*

En la *Tabla 6.25* se aprecia como la mejor tasa de acierto conseguida con la reducción de la dimensionalidad hecha por la red de neuronas, aplicando la función sigmoidal, es de 92,13%. Este valor se registra con 2 neuronas, 7 vecinos y la red de neuronas entrenada con 100 iteraciones.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
2	0.5212	0.5370	0.5380	0.5386
8	0.5190	0.5370	0.5326	0.5299
14	0.6435	0.6380	0.6636	0.6609
20	0.6554	0.6592	0.6685	0.6913
26	0.6560	0.6647	0.6886	0.6967
32	0.6630	0.6690	0.6826	0.6940
38	0.6658	0.6690	0.6832	0.6951

Tabla 6.26: *Resultados de KNN sobre los datos transformados por PCA en Spambase Rotados 8*

6.3.1.3. Experimentos añadiendo 50 atributos adicionales y rotando los datos

En este apartado se incluyen los resultados obtenidos añadiendo 50 atributos adicionales aleatorios al conjunto de datos *Spambase*, manteniendo los 57 atributos originales y su clase correspondiente. Teniendo en cuenta esto, el conjunto de datos utilizados en este caso para llevar a cabo las pruebas contienen 107 atributos en total y la clase.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.5310
3	0.5326
5	0.5674
7	0.5891

Tabla 6.27: *Resultados de KNN sobre los datos originales en Spambase Rotados 50*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.28* se puede apreciar la tasa media de acierto que proporciona KNN sobre la transformación lineal que realiza la capa oculta de la red.

	100 iteraciones				1.000 iteraciones			
Neuronas \ Vecinos	1	3	5	7	1	3	5	7
2	0.8822	0.9040	0.9087	0.8920	0.8866	0.9067	0.9102	0.9073
10	0.8508	0.8688	0.8742	0.8784	0.8499	0.8693	0.8754	0.8782
18	0.8192	0.8414	0.8486	0.8559	0.8202	0.8471	0.8538	0.8563
26	0.7938	0.8268	0.8386	0.8443	0.8016	0.8292	0.8378	0.8449
34	0.7815	0.8120	0.8237	0.8307	0.7858	0.8172	0.8276	0.8332
42	0.7864	0.8183	0.8328	0.8370	0.7842	0.8134	0.8235	0.8276
50	0.7850	0.8118	0.8216	0.8320	0.7816	0.8113	0.8178	0.8258

Tabla 6.28: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Spambase Rotados 50*

Observamos en la *Tabla 6.28* que con la transformación lineal y la red entrenada con 100 iteraciones se obtiene un máximo valor de 90,87%. Por otro lado, se observa que con la red entrenada con 1.000 iteraciones se alcanza un porcentaje de acierto máximo de 91,02%. Por este motivo, en este caso el mejor valor corresponde con un 91,02% y es producido con 2 neuronas en la capa oculta y 5 vecinos, habiendo ejecutado el entrenamiento con 1.000 iteraciones.

En la siguiente tabla se muestran los resultados obtenidos por KNN, utilizando la reducción de dimensionalidad producida por la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.8428	0.8585	0.8613	0.8502	0.7973	0.8148	0.8177	0.8155
10	0.7418	0.7551	0.7587	0.7570	0.7316	0.7509	0.7564	0.7599
18	0.7263	0.7421	0.7518	0.7549	0.7128	0.7363	0.7387	0.7401
26	0.6950	0.7216	0.7341	0.7409	0.7142	0.7321	0.7354	0.7373
34	0.6926	0.7095	0.7126	0.7173	0.6896	0.7119	0.7221	0.7212
42	0.6917	0.7017	0.7099	0.7090	0.7007	0.7183	0.7228	0.7223
50	0.6984	0.7208	0.7260	0.7299	0.6966	0.7099	0.7152	0.7159

Tabla 6.29: *Resultados de KNN sobre la transformación lineal normalizada en Spambase Rotados 50*

Comparando los mayores porcentajes de acierto de la tabla anterior se observa que con la red entrenada con 100 iteraciones se consigue un mejor resultado. Por ello, el mejor resultado de la transformación lineal normalizada corresponde con 86,13% y es registrado con 2 neuronas en la capa oculta y 5 vecinos.

En la *Tabla 6.30* se pueden observar los resultados proporcionados por KNN sobre la reducción de dimensionalidad hecha aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.8816	0.9040	0.9065	0.9070	0.8842	0.9099	0.9110	0.9200
10	0.8576	0.8715	0.8724	0.8723	0.8641	0.8786	0.8797	0.8804
18	0.8534	0.8665	0.8686	0.8668	0.8585	0.8771	0.8768	0.8768
26	0.8480	0.8614	0.8647	0.8654	0.8273	0.8449	0.8478	0.8500
34	0.8231	0.8349	0.8404	0.8421	0.8195	0.8362	0.8393	0.8405
42	0.8224	0.8387	0.8433	0.8436	0.8163	0.8343	0.8398	0.8400
50	0.7999	0.8172	0.8221	0.8280	0.8125	0.8329	0.8421	0.8441

Tabla 6.30: *Resultados de KNN aplicando la función sigmoidal en Spambase Rotados 50*

En el caso de la reducción de la dimensionalidad aplicando la función sigmoidal se observa que haciendo el entrenamiento con 100 iteraciones el máximo porcentaje de acierto alcanzado es de 90,70%, mientras que con 1.000 iteraciones el resultado es de 92%. Teniendo en cuenta estos valores, el mejor resultado que se produce es de 92% que corresponde con 2 neuronas en la capa oculta y 7 vecinos.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
2	0.5190	0.5179	0.5049	0.5223
10	0.5370	0.5375	0.5446	0.5538
18	0.5212	0.5495	0.5375	0.5413
26	0.5277	0.5201	0.5168	0.5217
34	0.5158	0.5255	0.5342	0.5326
42	0.5027	0.5299	0.5288	0.5440
50	0.4995	0.5109	0.5228	0.5299

Tabla 6.31: *Resultados de KNN sobre los datos transformados por PCA en Spambase Rotados 50*

6.3.2. Análisis de los resultados obtenidos en el dominio *Spambase*

6.3.2.1. Análisis de los resultados obtenidos en los experimentos sobre los datos originales

En primer lugar, se va a hacer un análisis de la calidad de la reducción de dimensionalidad llevada a cabo por la red de neuronas. Para realizar esto, se va a comparar la tasa de acierto obtenida por KNN sobre los datos originales con la tasa de acierto obtenida por KNN sobre los datos transformados con un menor número de atributos.

En la *Tabla 6.17* se muestra que el porcentaje de clasificación que se obtiene sobre los datos originales es de 89,57%. Este porcentaje es mejorado con los resultados logrados por la transformación lineal, ya que en este caso el mejor valor alcanzado con 12 neuronas en la capa oculta fue de 92,92%. Adicionalmente, cabe destacar que, como se comentó anteriormente, los datos originales del dominio *Spambase* contenían 57 atributos y con la reducción de la dimensionalidad producida por la transformación lineal únicamente se tendrían 12 atributos (12 neuronas en la capa oculta).

Sin embargo, con la transformación lineal normalizada el máximo porcentaje de acierto conseguido es de 88,26% con 28 neuronas en la capa oculta, por lo que no se consigue mejorar la tasa de acierto sobre los datos originales (89,57%). Cabe destacar que, aunque se obtiene un peor resultado, la diferencia entre ambos es muy pequeña y en este caso el conjunto de datos con dimensionalidad reducida tendría 28 atributos por los 57 que tiene el conjunto original.

Con la reducción de la dimensionalidad aplicando la función sigmoidal se registra un 93,20% con 8 neuronas, que también supera el resultado sobre los datos originales (89,57%). Nótese que esta reducción de la dimensionalidad llevada a cabo es muy rentable puesto que no solamente se mejora el resultado de clasificación de los datos originales (57 atributos), sino que el nuevo conjunto de datos transformados tendría únicamente 8 atributos. Además, cabe destacar que este resultado mejora significativamente el máximo valor ofrecido por PCA, ya que este proporciona un 91,85% y el nuevo conjunto de datos tendría 16 atributos.

Comparación entre los resultados de la transformación lineal y PCA

En la siguiente tabla se muestran los resultados de calcular la resta de la tasa de acierto obtenida en la reducción de la dimensionalidad realizada utilizando la transformación lineal de la capa oculta de la red y la tasa de acierto de PCA. Es importante destacar que la mejora obtenida será mayor cuanto mayor sean los valores de la tabla y, por el contrario, si se obtienen resultados negativos la solución propuesta empeorará los resultados de PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
4	0.0459	0.0497	0.0424	0.0352	0.0481	0.0500	0.0411	0.0360
8	0.0199	0.0263	0.0171	0.0194	0.0197	0.0275	0.0161	0.0191
12	0.0135	0.0069	0.0115	0.0063	0.0147	0.0105	0.0165	0.0140
16	0.0105	0.0118	0.0084	0.0054	0.0113	0.0134	0.0083	0.0067
20	0.0050	0.0043	0.0115	0.0085	0.0042	0.0042	0.0141	0.0107
24	0.0099	0.0099	0.0159	0.0153	0.0133	0.0111	0.0185	0.0159
28	0.0173	0.0140	0.0157	0.0150	0.0095	0.0095	0.0139	0.0117

Tabla 6.32: *Comparación entre los resultados de la transformación lineal y PCA en Spambase*

Como se puede observar en la tabla, los resultados con respecto a PCA se mejoran aunque no de una forma muy amplia. Por un lado, la mejora media que se produce con la transformación lineal y la red entrenada con 100 iteraciones es de un 1,69% y el valor máximo de mejora alcanzado es de 4,97%. Por otro lado, con la red de neuronas entrenada con 1.000 iteraciones se consigue una mejora media de 1,75% y un máximo de 5%.

En la siguiente gráfica se muestran los resultados obtenidos por la transformación lineal y PCA en el conjunto de datos *Spambase*. En la imagen se puede observar visualmente que con un número reducido de neuronas/componentes la mejora es amplia, mientras que PCA evoluciona mejor cuando aumenta el número de componentes.

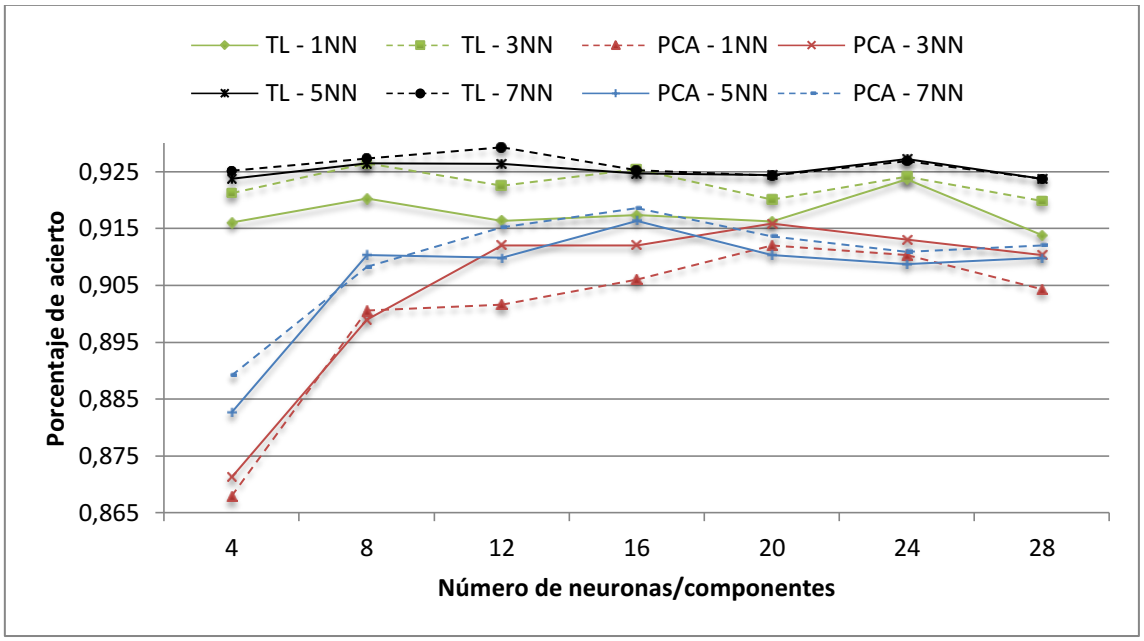

Ilustración 6.4: *Gráfica comparativa de la transformación lineal y PCA en Spambase*

Comparación entre los resultados de la transformación lineal normalizada y PCA

En la siguiente tabla se puede apreciar la comparación llevada a cabo entre los resultados obtenidos por KNN sobre la transformación lineal normalizada y PCA. Para hacer la comparación de los resultados obtenidos se ha calculado la resta entre ambas aproximaciones.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
4	-0.1686	-0.1582	-0.1606	-0.1641	-0.2171	-0.1938	-0.1911	-0.1827
8	-0.1244	-0.1063	-0.1108	-0.1053	-0.1456	-0.1510	-0.1512	-0.1457
12	-0.0752	-0.0733	-0.0657	-0.0702	-0.1213	-0.1057	-0.0766	-0.0770
16	-0.1153	-0.0908	-0.0945	-0.0895	-0.1086	-0.1030	-0.0990	-0.0972
20	-0.1008	-0.0908	-0.0798	-0.0797	-0.0750	-0.0693	-0.0547	-0.0528
24	-0.0866	-0.0734	-0.0652	-0.0648	-0.1039	-0.0959	-0.0736	-0.0739
28	-0.0477	-0.0468	-0.0311	-0.0294	-0.1016	-0.0925	-0.0834	-0.0784

Tabla 6.33: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Spambase*

En este caso se obtienen peores resultados que utilizando PCA en todas las situaciones, por lo que entrenando la red con 100 iteraciones se produce una pérdida media en la tasa de acierto del 9,17% con respecto a PCA. Además, como se puede apreciar en la tabla, incluso el mejor resultado que se obtiene es negativo perdiendo hasta un 2,94% de precisión.

Con la red de neuronas entrenada con 1.000 iteraciones también se obtienen resultados negativos. En la mejor situación se registra un 5,28% menos de acierto con respecto a PCA y el empeoramiento medio que se produce es de 11,15%.

Por tanto, en este caso no saldría rentable la reducción de la dimensionalidad con la transformación lineal normalizada.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.34* se puede observar la comparación entre los resultados de la reducción de la dimensionalidad obtenidos aplicando la función sigmoidal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
4	0.0501	0.0552	0.0469	0.0407	0.0492	0.0561	0.0466	0.0413
8	0.0182	0.0284	0.0201	0.0238	0.0220	0.0285	0.0171	0.0196
12	0.0156	0.0096	0.0150	0.0089	0.0158	0.0116	0.0144	0.0115
16	0.0149	0.0125	0.0100	0.0081	0.0147	0.0147	0.0120	0.0103
20	0.0051	0.0050	0.0154	0.0126	0.0042	0.0032	0.0095	0.0087
24	0.0075	0.0117	0.0168	0.0152	0.0098	0.0069	0.0128	0.0100
28	0.0143	0.0112	0.0139	0.0137	0.0115	0.0087	0.0121	0.0100

Tabla 6.34: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Spambase*

En la tabla anterior podemos comprobar que se mejoran todos los resultados de PCA, entrenando la red con 100 iteraciones la mejora media en la tasa de acierto que se logra

es del 1,86%, mientras que con la red entrenada con 1.000 iteraciones la mejora media que se alcanza es de 1,76%. Adicionalmente, se puede comprobar que en este caso se produce una mejora aplicando la función sigmoidal con respecto a los dos casos anteriores, ya que se mejoran ligeramente los resultados de la transformación lineal mientras que la mejora que se produce con respecto a la transformación lineal normalizada es muy superior.

En cuanto a los valores máximos de mejora producidos, podemos observar que con la red entrenada con 100 iteraciones se alcanza un 5,52% y con 1.000 iteraciones se obtiene un 5,61%.

6.3.2.2. Análisis de los resultados obtenidos en los experimentos añadiendo 8 atributos adicionales y rotando los datos

Como se observa en la *Tabla 6.22* el mejor resultado arrojado en la clasificación de los datos originales es de 68,45%.

En la *Tabla 6.23* se muestra que el mejor resultado logrado con la transformación lineal es de 92,12% con 2 neuronas en la capa oculta. Teniendo en cuenta estos valores, se puede apreciar que se produce una gran ganancia haciendo la reducción de la dimensionalidad, ya que, por un lado, se alcanza una mejora de 23,67% en la tasa de acierto y, por otro lado, el nuevo conjunto de datos tendría únicamente 2 atributos frente a los 65 atributos del conjunto de datos original.

En el caso de la transformación lineal normalizada el mejor resultado obtenido es de 84,08% con 38 neuronas en la capa oculta. Cabe destacar que también mejora la tasa de acierto de clasificación sobre los datos originales (68,45%) de una forma amplia, pero la mejora que se produce no es tan elevada como en el caso de la transformación lineal y, además, el conjunto de datos estaría formado por 38 atributos que es muy superior a los 2 atributos que serían necesarios en el caso anterior.

Por último, si observamos la reducción de la dimensionalidad llevada a cabo por la red de neuronas, aplicando la función sigmoidal, vemos que el mejor resultado obtenido es de 92,13% con 2 neuronas en la capa oculta. Nuevamente, se producen grandes mejoras realizando la reducción de la dimensionalidad, ya que se mejora hasta un 23,68% la tasa de acierto con respecto a la clasificación hecha sobre los datos originales y el nuevo conjunto de datos tendría 2 atributos.

Por ello, tanto la reducción de la dimensionalidad producida con la transformación lineal como aplicando la función sigmoidal son muy rentables. También, cabe destacar que el valor máximo alcanzado por PCA es 69,67% con 26 componentes que es un resultado mucho peor que los logrados con la red de neuronas.

En las siguientes tablas se muestra una comparación mucho más detallada de la reducción de la dimensionalidad realizada por la red de neuronas y PCA.

Comparación entre los resultados de la transformación lineal y PCA

En la *Tabla 6.35* se muestra la comparación hecha entre la tasa de acierto que se obtiene con la reducción de la dimensionalidad realizada con la transformación lineal y PCA. Como se comentó anteriormente, cuanto mayor sean los valores de la tabla, mayor será la mejora que se produce sobre PCA.

Vecinos / Elementos	100 iteraciones				1.000 iteraciones			
	1	3	5	7	1	3	5	7
2	0.3733	0.3815	0.3835	0.3835	0.3736	0.3782	0.3807	0.3815
8	0.3587	0.3632	0.3711	0.3796	0.3519	0.3550	0.3682	0.3741
14	0.2193	0.2526	0.2331	0.2387	0.2158	0.2499	0.2284	0.2343
20	0.1850	0.2116	0.2099	0.1933	0.1946	0.2224	0.2225	0.2058
26	0.1847	0.2069	0.1917	0.1897	0.1882	0.2087	0.1934	0.1892
32	0.1717	0.1980	0.1949	0.1875	0.1750	0.1962	0.1946	0.1878
38	0.1664	0.1918	0.1894	0.1831	0.1651	0.1937	0.1917	0.1870

Tabla 6.35: *Comparación entre los resultados de la transformación lineal y PCA en Spambase Rotados 8*

Como se puede apreciar en la tabla, se mejoran todos los resultados de PCA y, además, la mejora es significativa ya que la tasa de acierto media es superior en un 24,98% en el caso de 100 iteraciones y 25,03% con 1.000 iteraciones. Es importante destacar que los máximos que se obtienen son valores mucho más elevados que la media, concretamente del 38,35% y 38,15%. Esto es debido a que con la red de neuronas los mejores resultados se consiguen con un número reducido de neuronas y, sin embargo, con PCA a medida que aumentamos el número de componentes el rendimiento mejora.

En la *ilustración 6.5* se muestra la evolución de la transformación lineal y PCA en el dominio *Spambase rotados 8*. En la gráfica es posible observar la clara superioridad de los resultados ofrecidos por la transformación lineal.

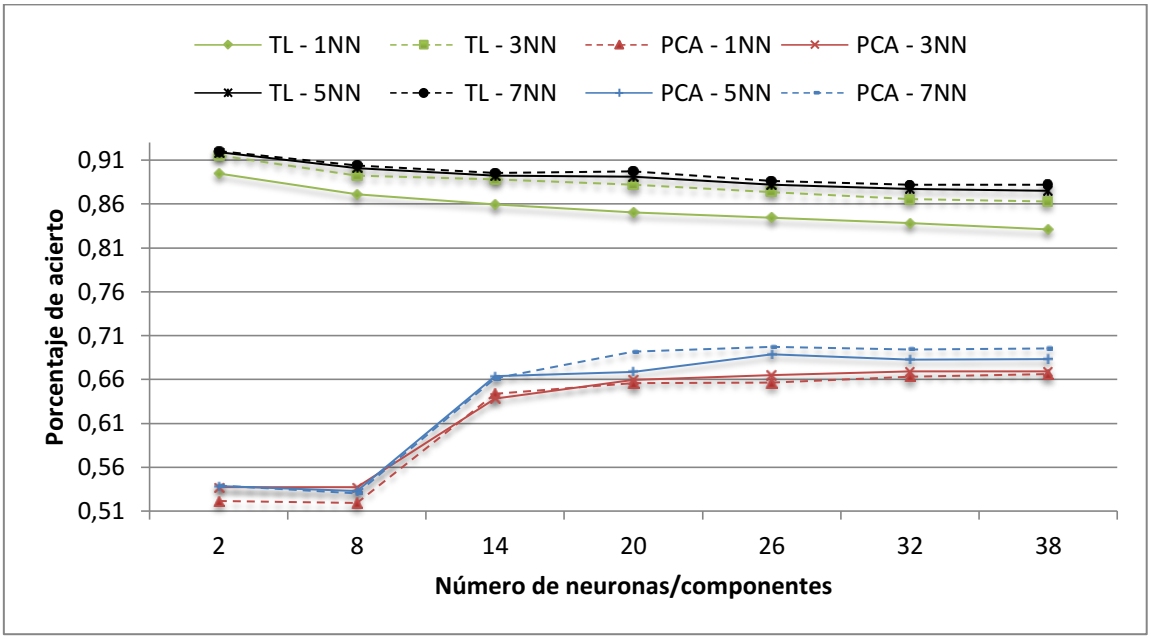

Ilustración 6.5: *Gráfica comparativa de la transformación lineal y PCA en Spambase rotados 8*

Teniendo en cuenta estos datos, en este caso sería muy rentable llevar a cabo la reducción de la dimensionalidad utilizando la red neuronal, ya que solo con 2 neuronas en la capa oculta se logran muchos mejores resultados que con cualquier número de componentes que usemos con PCA.

Comparación entre los resultados de la transformación lineal normalizada y PCA

A continuación, se muestra la comparación llevada a cabo de la reducción de la dimensionalidad empleando la transformación lineal normalizada y PCA. Para obtener los valores de la tabla se ha calculado la resta de los resultados obtenidos con la transformación lineal normalizada y los resultados de PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.2074	0.2010	0.2041	0.2011	0.1530	0.1452	0.1438	0.1521
8	0.2426	0.2538	0.2724	0.2812	0.2430	0.2507	0.2665	0.2722
14	0.1225	0.1605	0.1468	0.1568	0.0989	0.1334	0.1136	0.1233
20	0.1370	0.1668	0.1650	0.1446	0.1193	0.1422	0.1451	0.1317
26	0.1152	0.1304	0.1155	0.1135	0.1229	0.1429	0.1318	0.1269
32	0.1162	0.1339	0.1308	0.1207	0.1110	0.1333	0.1279	0.1209
38	0.2074	0.2010	0.2041	0.2011	0.1191	0.1538	0.1519	0.1457

Tabla 6.36: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Spambase Rotados 8*

Como se puede apreciar en la tabla anterior, se mejoran en todos los casos los resultados de PCA, es decir, no existen resultados negativos. Con 100 iteraciones la mejora media que se produce es del 16,83% y el valor máximo es 28,12% con 8 neuronas y 7 vecinos. Por otro lado, con 1.000 iteraciones se consigue una mejora media de 15,08% y el valor máximo que se alcanza es 27,22%.

Como se puede comprobar, aunque se mejoran los resultados de PCA, los resultados que se obtienen con la transformación lineal son mejores que los resultados de la transformación lineal normalizada.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.37* se pueden observar los resultados obtenidos en la comparación de la reducción de la dimensionalidad realizada por la red de neuronas, aplicando la función sigmoidal, y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.3721	0.3806	0.3828	0.3827	0.3729	0.3797	0.3815	0.3805
8	0.3720	0.3756	0.3838	0.3871	0.3704	0.3736	0.3805	0.3828
14	0.2476	0.2710	0.2472	0.2490	0.2388	0.2603	0.2377	0.2415
20	0.2283	0.2383	0.2313	0.2102	0.2167	0.2283	0.2227	0.2021
26	0.2270	0.2378	0.2178	0.2086	0.2283	0.2332	0.2115	0.2035
32	0.2088	0.2171	0.2095	0.1992	0.2090	0.2183	0.2097	0.1999
38	0.1908	0.2088	0.1983	0.1892	0.2002	0.2168	0.2069	0.1997

Tabla 6.37: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Spambase Rotados 8*

Nuevamente, se mejoran todos los resultados de PCA de forma significativa, el porcentaje de mejora con 100 iteraciones es del 26,69%, mientras que con 1.000 iteraciones se obtiene una mejora media de 26,45%. Por otro lado, merece la pena

destacar que en ambos casos se logran unos valores máximos muy elevados con 8 neuronas y 7 vecinos, estos porcentajes de acierto son 38,71% y 38,28%.

6.3.2.3. Análisis de los resultados obtenidos en los experimentos añadiendo 50 atributos adicionales y rotando los datos

En este caso se registra un 58,91% de acierto en la clasificación de los datos originales (véase *Tabla 6.27*). De forma análoga al caso anterior, los resultados obtenidos sobre los datos originales son mejorados de una forma muy amplia por la transformación lineal y aplicando la función sigmoidal, ya que se alcanza un 91,02% y un 92% de acierto (en ambos casos con 2 neuronas en la capa oculta), respectivamente. Por tanto, sería muy rentable realizar la reducción de la dimensionalidad con cualquiera de las dos aproximaciones puesto que la mejora que se produce es superior a un 30% en ambos casos. Adicionalmente, es importante destacar que el conjunto de datos original tiene 107 atributos y con el nuevo conjunto de datos, habiendo hecho la reducción de la dimensionalidad, se tendrían únicamente 2 atributos.

Con la transformación lineal normalizada también se produce una mejora bastante grande, ya que se obtiene un valor máximo de 86,13% con 2 neuronas. Como se puede observar, también se supera la tasa de acierto producida en los datos originales (58,91%) utilizando menos atributos.

En la *Tabla 6.31* se observa que el porcentaje de acierto máximo registrado por PCA es de 55,38% con 10 componentes. Merece la pena destacar que no solo ofrece peores resultados que la red de neuronas, sino que tampoco es capaz de mejorar el resultado de clasificación de los datos originales (58,91%).

Comparación entre los resultados de la transformación lineal y PCA

En la siguiente tabla se muestra la comparación de la reducción de la dimensionalidad producida con la transformación lineal de la red neuronal y PCA. Dicha tabla ha sido obtenida aplicando la resta de la tasa media de acierto que se obtiene con la transformación lineal y la tasa media de PCA. Por tanto, la mejora será mayor cuanto más grandes sean los valores de la tabla.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.3632	0.3861	0.4038	0.3697	0.3676	0.3888	0.4053	0.3850
10	0.3138	0.3313	0.3296	0.3246	0.3129	0.3318	0.3308	0.3244
18	0.2980	0.2919	0.3111	0.3146	0.2990	0.2976	0.3163	0.3150
26	0.2661	0.3067	0.3218	0.3226	0.2739	0.3091	0.3210	0.3232
34	0.2657	0.2865	0.2895	0.2981	0.2700	0.2917	0.2934	0.3006
42	0.2837	0.2884	0.3040	0.2930	0.2815	0.2835	0.2947	0.2836
50	0.2855	0.3009	0.2988	0.3021	0.2821	0.3004	0.2950	0.2959

Tabla 6.38: *Comparación entre los resultados de la transformación lineal y PCA en Spambase Rotados50*

Se puede comprobar en la tabla como se produce una mejora muy grande con respecto a PCA, con 1.000 iteraciones el porcentaje medio de mejora es del 31,34% y el valor máximo de mejora que se alcanza es del 40,53%. Se obtiene un resultado similar

entrenando la red con 100 iteraciones, ya que la mejora media lograda es de 31,25% y un valor máximo de 40,38%.

En la siguiente ilustración se muestran de forma gráfica los resultados proporcionados por la transformación lineal y PCA. De forma análoga a los casos anteriores, en dicha gráfica es posible apreciar que PCA ofrece unos resultados muy inferiores comparado con la transformación lineal.

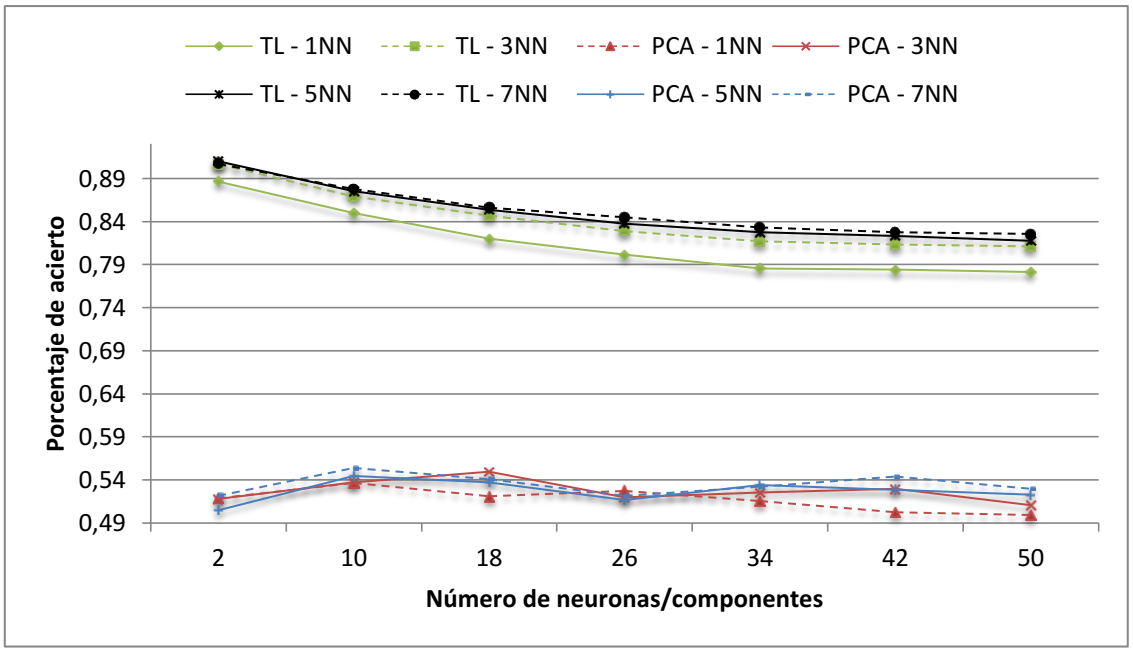

Ilustración 6.6: *Gráfica comparativa de la transformación lineal y PCA en Spambase rotados 50*

Comparación entre los resultados de la transformación lineal normalizada y PCA

En la *Tabla 6.39* se muestra la comparación hecha entre la transformación lineal normalizada y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.3238	0.3406	0.3564	0.3279	0.2783	0.2969	0.3128	0.2932
10	0.2048	0.2176	0.2141	0.2032	0.1946	0.2134	0.2118	0.2061
18	0.2051	0.1926	0.2143	0.2136	0.1916	0.1868	0.2012	0.1988
26	0.1673	0.2015	0.2173	0.2192	0.1865	0.2120	0.2186	0.2156
34	0.1768	0.1840	0.1784	0.1847	0.1738	0.1864	0.1879	0.1886
42	0.1890	0.1718	0.1811	0.1650	0.1980	0.1884	0.1940	0.1783
50	0.1989	0.2099	0.2032	0.2000	0.1971	0.1990	0.1924	0.1860

Tabla 6.39: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Spambase Rotados 50*

En este caso, nuevamente, se mejoran todos los resultados de PCA, con 1.000 iteraciones la tasa de acierto media alcanzada es superior a PCA en un 21,03% y el valor máximo registrado es de 31,28%. Por otro lado, entrenando la red con 100 iteraciones se logra una mejora media de 31,25% y un valor máximo de 35,64%.

Merece la pena destacar que el resultado de la reducción de la dimensionalidad realizada por la red de neuronas parece mejor a medida que aumenta el número de atributos irrelevantes en el dominio, ya que en este caso (con 50 atributos con valores aleatorios) la mejora es del 21,03% y, sin embargo, con los datos originales en la comparación entre transformación lineal normalizada y PCA se empeoraba el resultado en 9,17%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.40* se muestra la comparación llevada a cabo entre la tasa de acierto media obtenida en la reducción de la dimensionalidad producida por la red de neuronas, aplicando la función sigmoidal, y PCA. Los valores de la tabla han sido obtenidos calculando la resta entre los resultados aplicando la función sigmoidal y PCA

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.3626	0.3861	0.4016	0.3847	0.3652	0.3920	0.4061	0.3977
10	0.3206	0.3340	0.3278	0.3185	0.3271	0.3411	0.3351	0.3266
18	0.3322	0.3170	0.3311	0.3255	0.3373	0.3276	0.3393	0.3355
26	0.3203	0.3413	0.3479	0.3437	0.2996	0.3248	0.3310	0.3283
34	0.3073	0.3094	0.3062	0.3095	0.3037	0.3107	0.3051	0.3079
42	0.3197	0.3088	0.3145	0.2996	0.3136	0.3044	0.3110	0.2960
50	0.3004	0.3063	0.2993	0.2981	0.3130	0.3220	0.3193	0.3142

Tabla 6.40: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Spambase Rotados 50*

De nuevo los resultados que se logran son mejores que PCA en todos los casos, ya que no hay ningún valor negativo en la tabla. Entrenando la red con 100 iteraciones se produce una mejora media de 32,76% y un valor máximo de 40,16%. Valores muy similares son registrados con la red entrenada con 1.000 iteraciones puesto que la tasa de acierto media que se mejora con respecto a PCA es del 32,98% y el valor máximo que se obtiene es del 40,61%.

6.4. Dominio *German*

El domino conocido con el nombre *German* ha sido donado por Hans Hofmann, profesor de la universidad de Hamburgo, y contiene una base de datos de usuarios de una cuenta bancaria que son clasificados como buenos o malos clientes. El conjunto de datos está formado por 1.000 instancias y 24 atributos, algunos de estos atributos son los siguientes: Estado de la cuenta corriente existente, duración (en meses), historia de crédito, propósito, cantidad de crédito, cuenta de ahorros/bonos, periodo de tiempo que lleva en su presente empleo, tasa de desembolso en porcentaje del ingreso disponible, estado personal y sexo, otros deudores/aval, periodo de tiempo que lleva en su actual residencia, propiedad, edad del usuario (en años), otros planes de cuotas, alojamiento, número de créditos en el banco, trabajo, número de personas en el núcleo familiar, teléfono y trabajo externo.

6.4.1. Experimentos realizados sobre el dominio *German*

6.4.1.1. Experimentos sobre los datos originales

En este apartado se detallan los experimentos ejecutados sobre el conjunto de datos llamado *German*. Por tanto, como se ha comentado anteriormente, los datos están formados por 24 atributos y una clase de salida que contiene si el usuario del banco es bueno o malo.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.6900
3	0.7150
5	0.6975
7	0.6800

Tabla 6.41: *Resultados de KNN sobre los datos originales en German*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.42* se muestra la tasa media de acierto arrojada por KNN sobre la reducción de la dimensionalidad realizada por la transformación lineal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.6645	0.6955	0.6975	0.7010	0.6783	0.7038	0.7104	0.7167
4	0.6854	0.7057	0.7086	0.7168	0.6710	0.6885	0.7045	0.7075
6	0.6855	0.7115	0.7205	0.7165	0.6850	0.7150	0.7165	0.7175
8	0.6910	0.7130	0.7265	0.7200	0.7050	0.7175	0.7240	0.7205
10	0.7054	0.7171	0.7171	0.7229	0.6814	0.7182	0.7246	0.7304
12	0.6858	0.7079	0.7242	0.7233	0.7088	0.7180	0.7192	0.7321
14	0.7035	0.7253	0.7318	0.7295	0.7055	0.7225	0.7285	0.7240

Tabla 6.42: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en German*

En la *Tabla 6.42* el mejor resultado que se registra con la red entrenada con 100 iteraciones es de 73,18%, mientras que con 1.000 iteraciones el mejor resultado obtenido es 73,21%. Por tanto, el mejor resultado de reducción de dimensionalidad con la transformación lineal es 73,21% que se ofrece con 12 neuronas en la capa oculta y 7 vecinos.

En la siguiente tabla se pueden observar los resultados obtenidos por KNN sobre la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.6660	0.6900	0.7005	0.7035	0.6700	0.6996	0.7021	0.7104
4	0.6529	0.6864	0.6986	0.7075	0.6755	0.6925	0.6990	0.6995
6	0.6715	0.7020	0.7145	0.7105	0.6950	0.7045	0.7110	0.7100
8	0.6840	0.7175	0.7085	0.7110	0.6860	0.7175	0.7175	0.7205
10	0.6929	0.7117	0.7208	0.7208	0.6879	0.7054	0.7161	0.7214
12	0.6958	0.7083	0.7217	0.7258	0.6954	0.7075	0.7167	0.7117
14	0.6883	0.7113	0.7213	0.7203	0.6875	0.7105	0.7120	0.7240

Tabla 6.43: *Resultados de KNN sobre la transformación lineal normalizada en German*

En la *Tabla 6.43* se observa que con la red de neuronas entrenada con 100 iteraciones el mejor resultado que se alcanza es 72,58%, este resultado mejora el máximo proporcionado con la red entrenada con 1.000 iteraciones. Teniendo en cuenta estos datos, el máximo valor con la transformación lineal normalizada es 72,58% y se registra con 12 neuronas en la capa oculta y 7 vecinos.

En la siguiente tabla se muestra la tasa media de acierto obtenida por KNN sobre la reducción de la dimensionalidad hecha por la red de neuronas, aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.6775	0.6980	0.6995	0.7045	0.6888	0.7038	0.7121	0.7138
4	0.6646	0.6900	0.6946	0.6968	0.6545	0.6960	0.7015	0.7065
6	0.6595	0.6995	0.7025	0.7070	0.6660	0.6875	0.6925	0.6930
8	0.6780	0.7000	0.7080	0.7080	0.6675	0.6880	0.7030	0.7020
10	0.6763	0.6958	0.7038	0.7088	0.6836	0.7000	0.7057	0.7100
12	0.6867	0.7013	0.7063	0.7104	0.6783	0.6983	0.7075	0.7063
14	0.6838	0.7008	0.7075	0.7115	0.6990	0.7110	0.7110	0.7125

Tabla 6.44: *Resultados de KNN aplicando la función sigmoidal en German*

Por último, podemos observar en la *Tabla 6.44* que aplicando la función sigmoidal y entrenando la red con 100 iteraciones el mejor resultado obtenido es de 71,15%, y en el caso de 1.000 iteraciones se logra un valor máximo de 71,38% que mejora el resultado anterior. Por tanto, el mejor resultado aplicando la función sigmoidal es 71,38% y se registra con 2 neuronas en la capa oculta y 7 vecinos.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
2	0.6550	0.6550	0.6575	0.6575
4	0.6275	0.6450	0.6625	0.6525
6	0.6300	0.6825	0.6825	0.6825
8	0.6725	0.7275	0.7300	0.7250
10	0.6825	0.7300	0.7325	0.7325
12	0.6725	0.7125	0.7275	0.7225
14	0.6825	0.7325	0.7350	0.7300

Tabla 6.45: *Resultados de KNN sobre los datos transformados por PCA en German*

6.4.1.2. Experimentos añadiendo 8 atributos adicionales y rotando los datos

En el presente apartado se muestran los resultados de los experimentos hechos sobre el conjunto de datos llamado *German* al que se le han añadido 8 atributos adicionales calculados de forma aleatoria en el rango [0,1]. Además de esto, se ha realizado posteriormente una rotación de los datos. Por tanto, en este caso el conjunto de datos está formado por 32 atributos (los 24 atributos originales se mantienen y se han añadido 8 adicionales) y la clase.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.6450
3	0.6850
5	0.7050
7	0.7100

Tabla 6.46: *Resultados de KNN sobre los datos originales en German Rotados 8*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la siguiente tabla se muestra la tasa de acierto media obtenida con KNN sobre la transformación lineal que hace la capa oculta de la red.

	100 iteraciones				1.000 iteraciones			
Neuronas \ Vecinos	1	3	5	7	1	3	5	7
2	0.6738	0.6892	0.7121	0.7154	0.6825	0.7100	0.7140	0.7175
4	0.6840	0.7115	0.7275	0.7320	0.6838	0.7063	0.7125	0.7217
6	0.6650	0.7000	0.7130	0.7170	0.6525	0.6875	0.7040	0.7140
8	0.6690	0.6890	0.7100	0.7165	0.6865	0.7125	0.7275	0.7235
10	0.6771	0.7017	0.7183	0.7229	0.6688	0.6958	0.7142	0.7163
12	0.6550	0.6958	0.7092	0.7113	0.6760	0.7175	0.7240	0.7305
14	0.6640	0.6935	0.7040	0.7135	0.6760	0.7090	0.7125	0.7085

Tabla 6.47: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en German Rotados 8*

En la *Tabla 6.47*, que contiene los resultados de la transformación lineal, se observa que ejecutando el entrenamiento con 100 iteraciones el mejor resultado alcanzado es de 73,20%. Con la red entrenada con 1.000 iteraciones no se consigue mejorar este resultado, ya que el máximo obtenido en este caso es de 73,05%. Por ello, el mejor resultado logrado con la transformación lineal es de 73,20% con 4 neuronas y 7 vecinos.

En la *Tabla 6.48* se muestran los resultados de clasificación obtenidos por KNN sobre la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.6704	0.7000	0.7146	0.7188	0.6795	0.7005	0.7160	0.7185
4	0.6670	0.7115	0.7165	0.7165	0.6692	0.7021	0.7063	0.7117
6	0.6710	0.6950	0.7095	0.7130	0.6590	0.6885	0.7100	0.7135
8	0.6450	0.6925	0.7025	0.7145	0.6785	0.6945	0.7035	0.7100
10	0.6763	0.7000	0.7171	0.7246	0.6663	0.7079	0.7096	0.7108
12	0.6504	0.6896	0.7029	0.7117	0.6585	0.6830	0.6885	0.7005
14	0.6595	0.6935	0.7035	0.7115	0.6675	0.6980	0.7105	0.7135

Tabla 6.48: *Resultados de KNN sobre la transformación lineal normalizada en German Rotados 8*

Comparando los valores máximos de la *Tabla 6.48*, se puede comprobar como en el caso de la transformación lineal normalizada también se obtiene el mejor resultado con la red entrenada con 100 iteraciones. Este máximo valor es conseguido con 10 neuronas y corresponde con 72,46%.

En la siguiente tabla se muestran los resultados de clasificación obtenidos por KNN sobre la reducción de la dimensionalidad producida por la red de neuronas aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.6988	0.7133	0.7154	0.7171	0.6805	0.7030	0.7140	0.7165
4	0.6760	0.6925	0.6985	0.7025	0.6825	0.6996	0.7004	0.7029
6	0.6660	0.6760	0.6800	0.6780	0.6650	0.6845	0.6820	0.6880
8	0.6540	0.6820	0.6770	0.6785	0.6845	0.6950	0.6935	0.6990
10	0.6729	0.6933	0.6967	0.7000	0.6742	0.6896	0.6921	0.6925
12	0.6708	0.6838	0.6850	0.6842	0.6645	0.6785	0.6885	0.6895
14	0.6870	0.6945	0.6900	0.6900	0.6565	0.6705	0.6790	0.6735

Tabla 6.49: *Resultados de KNN aplicando la función sigmoidal en German Rotados 8*

Los resultados de la reducción de la dimensionalidad llevada a cabo por la red de neuronas aplicando la función sigmoidal se muestran en la *Tabla 6.49*, en ella podemos comprobar que el máximo porcentaje de acierto ofrecido es de 71,71%. Este resultado es ligeramente mejor que el máximo porcentaje arrojado con 1.000 iteraciones. Por este motivo, la mejor tasa de acierto alcanzada aplicando la función sigmoidal es de 71,71% con 2 neuronas en la capa oculta.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
2	0.6175	0.6400	0.6450	0.6325
4	0.6475	0.6675	0.6875	0.7025
6	0.6500	0.6750	0.6550	0.6650
8	0.6625	0.6900	0.6875	0.6850
10	0.6825	0.7025	0.7200	0.7000
12	0.6225	0.7150	0.7050	0.7000
14	0.6300	0.6875	0.6950	0.7050

Tabla 6.50: *Resultados de KNN sobre los datos transformados por PCA en German Rotados 8*

6.4.1.3. Experimentos añadiendo 24 atributos adicionales y rotando los datos

En este apartado se incluyen los resultados obtenidos añadiendo 24 atributos adicionales aleatorios al conjunto de datos original, manteniendo los 24 atributos originales y su clase correspondiente. Teniendo en cuenta esto, el conjunto de datos utilizados en este caso para realizar las pruebas contienen 48 atributos en total y la clase.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.6200
3	0.6750
5	0.6775
7	0.6925

Tabla 6.51: *Resultados de KNN sobre los datos originales en German Rotados 24*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.52* se muestran los resultados obtenidos por KNN sobre la transformación lineal.

	100 iteraciones				1.000 iteraciones			
Neuronas \ Vecinos	1	3	5	7	1	3	5	7
2	0.6650	0.6940	0.6955	0.7040	0.6689	0.6889	0.7036	0.7029
6	0.6720	0.6910	0.6890	0.6815	0.6650	0.6820	0.6945	0.6990
10	0.6710	0.7010	0.7075	0.7080	0.6755	0.6970	0.7045	0.7050
14	0.6830	0.6960	0.7015	0.7125	0.6766	0.6913	0.6972	0.7044
18	0.6740	0.7040	0.7050	0.7175	0.6592	0.6883	0.6983	0.7054
20	0.6686	0.6893	0.7004	0.7089	0.6579	0.6871	0.7050	0.7046
24	0.6654	0.6925	0.7007	0.7100	0.6640	0.6985	0.7015	0.7040

Tabla 6.52: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en German Rotados 24*

Como se observa en la *Tabla 6.52*, con la reducción de la dimensionalidad calculada por la transformación lineal y la red entrenada con 100 iteraciones el mejor resultado que se consigue es de 71,75%. Por otro lado, se observa que el mayor porcentaje logrado habiendo realizado el entrenamiento de la red con 1.000 iteraciones es de 70,54%. Por tanto, el mejor resultado de la transformación lineal corresponde con 71,75% y es registrado con 18 neuronas en la capa oculta.

A continuación, se muestran los resultados obtenidos por KNN sobre la reducción de la dimensionalidad hecha por la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.6475	0.6870	0.6965	0.6980	0.6707	0.6829	0.6911	0.6986
6	0.6570	0.6825	0.6745	0.6845	0.6785	0.7050	0.7085	0.7060
10	0.6715	0.7030	0.7135	0.7150	0.6805	0.6990	0.7110	0.7185
14	0.6710	0.7035	0.7120	0.7110	0.6741	0.6941	0.7028	0.7072
18	0.6695	0.6935	0.7095	0.7170	0.6683	0.7004	0.7013	0.7192
20	0.6679	0.7036	0.7104	0.7068	0.6596	0.6858	0.7025	0.7083
24	0.6543	0.6936	0.7139	0.7229	0.6650	0.6950	0.7170	0.7210

Tabla 6.53: *Resultados de KNN sobre la transformación lineal normalizada en German Rotados 24*

En el caso de la transformación lineal normalizada el mejor resultado ofrecido es de 72,29%. Este valor es obtenido comparando la mayor tasa de acierto alcanzada con la red entrenada con 100 iteraciones y la alcanzada con la red entrenada con 1.000 iteraciones. Además, es importante destacar que este resultado se produce con 24 neuronas y 7 vecinos.

En la siguiente tabla se pueden observar los resultados obtenidos por KNN sobre la reducción de dimensionalidad realizada aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
2	0.6525	0.6770	0.7020	0.7045	0.6707	0.6986	0.6989	0.7054
6	0.6705	0.6905	0.6960	0.6965	0.6860	0.6925	0.6980	0.7030
10	0.6870	0.7020	0.7025	0.7030	0.6845	0.7025	0.7025	0.7035
14	0.6610	0.6680	0.6790	0.6800	0.6847	0.6959	0.7041	0.7019
18	0.6795	0.7025	0.7040	0.7045	0.6788	0.6888	0.6954	0.6921
20	0.6850	0.7029	0.7068	0.7061	0.6763	0.6896	0.6929	0.6938
24	0.6704	0.6825	0.6857	0.6918	0.6780	0.6890	0.6925	0.6925

Tabla 6.54: *Resultados de KNN aplicando la función sigmoidal en German Rotados 24*

Por último, se observa que el mejor resultado logrado con la reducción de la dimensionalidad con la red de neuronas, aplicando la función sigmoidal y entrenando la red con 100 iteraciones, es de 70,68%, mientras que el mejor resultado registrado entrenando la red con 1.000 iteraciones es de 70,54%. Por ello, la mejor tasa ofrecida aplicando la función sigmoidal es de 70,68% con 20 neuronas en la capa oculta.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
2	0.6400	0.6700	0.6525	0.6650
6	0.6700	0.6775	0.6875	0.7150
10	0.6500	0.6650	0.6625	0.6800
14	0.6250	0.6600	0.6750	0.7000
18	0.6350	0.6450	0.6625	0.6700
20	0.6150	0.6700	0.6725	0.6750
24	0.6000	0.6525	0.6800	0.6775

Tabla 6.55: *Resultados de KNN sobre los datos transformados por PCA en German Rotados 24*

6.4.2. Análisis de los resultados obtenidos en el dominio *German*

6.4.2.1. Análisis de los resultados obtenidos en los experimentos sobre los datos originales

En la *Tabla 6.41* se muestra como la mayor tasa de acierto conseguida por KNN sobre el conjunto de datos original es de 71,50%.

Como se comentó anteriormente, la mejor tasa de acierto alcanzada en la reducción de la dimensionalidad realizada con la transformación lineal es de 73,21% con 12 neuronas en la capa oculta. Este resultado mejora ligeramente el resultado conseguido sobre los datos originales (71,50%). Un resultado similar es obtenido con la transformación lineal normalizada, ya que se alcanza un 72,58% con 12 neuronas en la capa oculta, mejorando también la tasa de acierto en los datos originales.

El mejor resultado conseguido aplicando la función sigmoidal es de 71,38% con 2 neuronas en la capa oculta. En este caso se empeora el porcentaje de acierto obtenido sobre los datos originales (71,50%), pero merece la pena destacar que la diferencia es muy pequeña (tan solo de 0,12%) y este resultado se produce con un conjunto de datos con 2 neuronas frente a los 24 atributos que tiene el conjunto de datos original.

Adicionalmente, en este conjunto de datos el mejor resultado que se registra con PCA es de 73,50%. Como se puede observar, este resultado mejora tanto el porcentaje de acierto sobre los datos originales (71,50%) como los porcentajes de acierto logrados con la red de neuronas (73,21%, 72,58% y 71,38%), pero esta mejora no es en ningún caso significativa.

A continuación, se va a mostrar una comparación más profunda de los resultados ofrecidos en la clasificación con la reducción de la dimensionalidad realizada por la red de neuronas y PCA.

Comparación entre los resultados de la transformación lineal y PCA

En la siguiente tabla podemos observar la comparación calculada entre los resultados de la transformación lineal y PCA.

Elementos \ Vecinos	100 iteraciones				1.000 iteraciones			
	1	3	5	7	1	3	5	7
2	0.0095	0.0405	0.0400	0.0435	0.0233	0.0488	0.0529	0.0592
4	0.0579	0.0607	0.0461	0.0643	0.0435	0.0435	0.0420	0.0550
6	0.0555	0.0290	0.0380	0.0340	0.0550	0.0325	0.0340	0.0350
8	0.0185	-0.0145	-0.0035	-0.0050	0.0325	-0.0100	-0.0060	-0.0045
10	0.0229	-0.0129	-0.0154	-0.0096	-0.0011	-0.0118	-0.0079	-0.0021
12	0.0133	-0.0046	-0.0033	0.0008	0.0363	0.0055	-0.0083	0.0096
14	0.0210	-0.0072	-0.0032	-0.0005	0.0230	-0.0100	-0.0065	-0.0060

Tabla 6.56: *Comparación entre los resultados de la transformación lineal y PCA en German*

En la tabla comparativa se muestra que se producen algunos resultados negativos, es decir, en 22 casos se empeoran los resultados de PCA. Sin embargo, en los 34 casos restantes se logran mejores resultados con la transformación lineal. Nuevamente, aunque hay varios casos negativos, se sigue produciendo una mejora ya que haciendo la media se obtienen valores positivos, por un lado, con 100 iteraciones se consigue un 1,84% y, por otro lado, con 1.000 iteraciones se alcanza un 1,99% de mejora.

Además, cabe destacar que entrenando la red con 100 iteraciones la mejora máxima registrada es de 6,43%, mientras que con 1.000 iteraciones el máximo alcanzado es 5,92%.

Los resultados de la transformación lineal y PCA en el dominio *German* se muestran gráficamente en las siguientes ilustraciones. En la *ilustración 6.7* se muestran los resultados para *k* igual a 1 y 3, mientras que en la *ilustración 6.8* se pueden apreciar los resultados para *k* igual a 5 y 7. Se puede observar que en este dominio los resultados están más igualados que en casos anteriores e incluso en algunas ocasiones se obtiene un resultado más elevado con PCA.

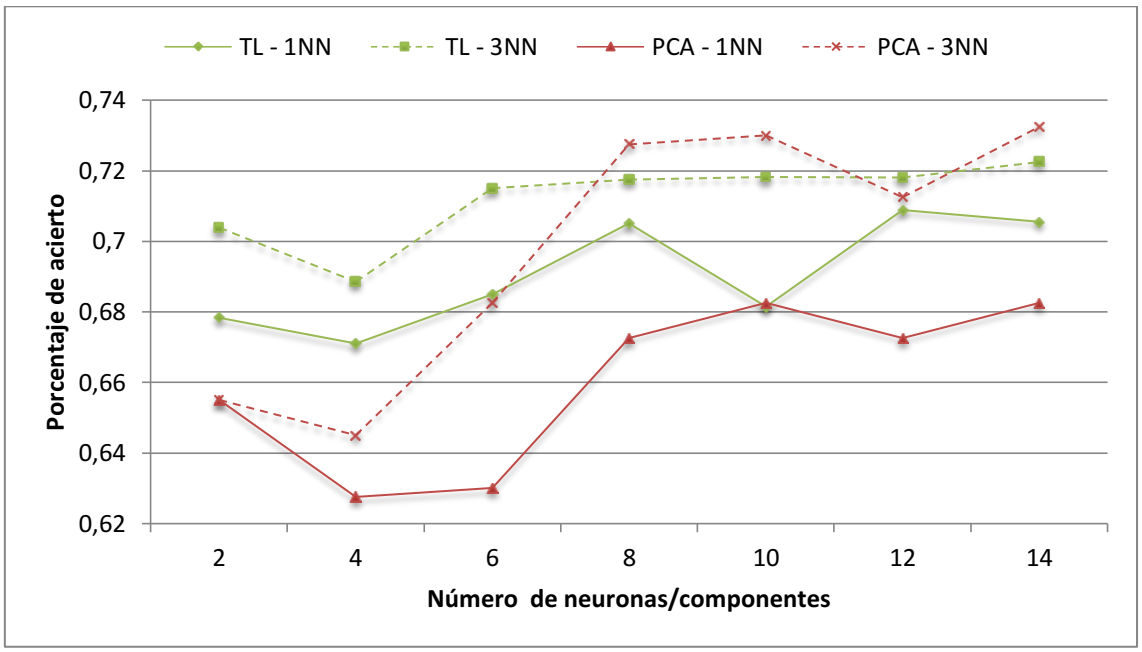

Ilustración 6.7: *Gráfica comparativa de la transformación lineal y PCA en German para k=1 y k=3*

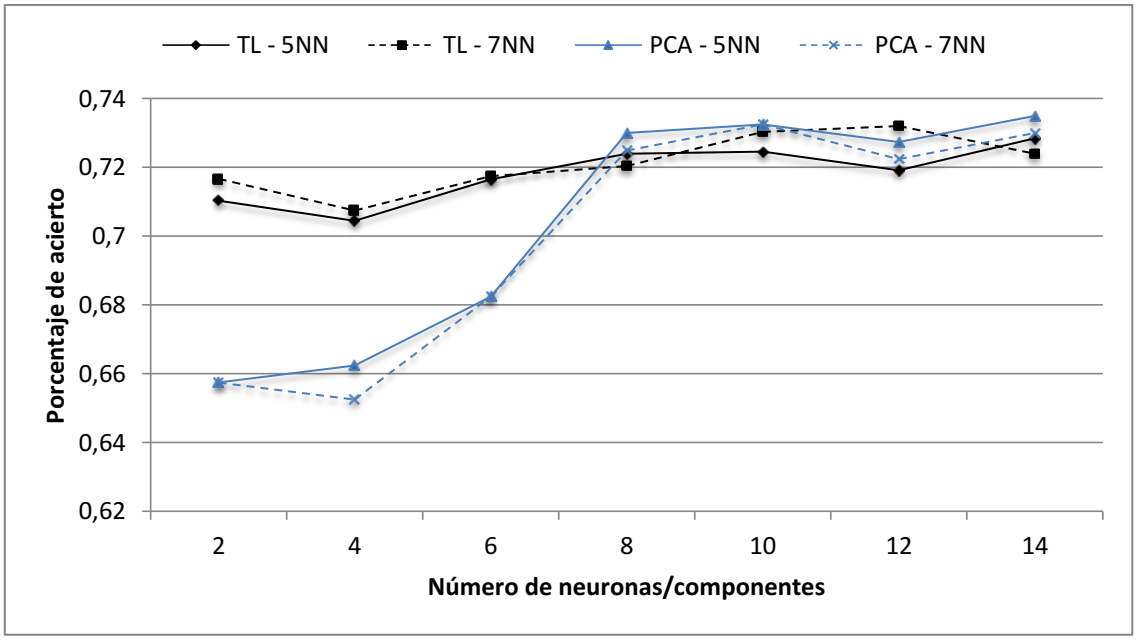

Ilustración 6.8: *Gráfica comparativa de la transformación lineal y PCA en German para k=5 y k=7*

Comparación entre los resultados de la transformación lineal normalizada y PCA

En la siguiente tabla se muestra la comparación de la transformación lineal normalizada y PCA. Para obtener los valores de la tabla se ha calculado la resta de los resultados producidos con la transformación lineal normalizada y los resultados de PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.0110	0.0350	0.0430	0.0460	0.0150	0.0446	0.0446	0.0529
4	0.0254	0.0414	0.0361	0.0550	0.0480	0.0475	0.0365	0.0470
6	0.0415	0.0195	0.0320	0.0280	0.0650	0.0220	0.0285	0.0275
8	0.0115	-0.0100	-0.0215	-0.0140	0.0135	-0.0100	-0.0125	-0.0045
10	0.0104	-0.0183	-0.0117	-0.0117	0.0054	-0.0246	-0.0164	-0.0111
12	0.0233	-0.0042	-0.0058	0.0033	0.0229	-0.0050	-0.0108	-0.0108
14	0.0058	-0.0212	-0.0137	-0.0097	0.0050	-0.0220	-0.0230	-0.0060

Tabla 6.57*: Comparación entre los resultados de la transformación lineal normalizada y PCA en German*

En la tabla anterior se muestra que los resultados se empeoran en 13 ocasiones y en el resto se produce una mejora. Por tanto, entrenando la red con 100 iteraciones el valor máximo de mejora que se alcanza es de 5,50% y la mejora media que se produce es de 1,17%. Resultados similares son ofrecidos con la red entrenada con 1.000 iteraciones, ya que se consigue un 1,32% de mejora media y el valor máximo corresponde con 6,50%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.58* se muestran los resultados de aplicar la resta de la tasa de acierto obtenida en la reducción de la dimensionalidad realizada aplicando la función sigmoidal y la tasa de acierto obtenida con PCA.

Vecinos / Elementos	100 iteraciones				1.000 iteraciones			
	1	3	5	7	1	3	5	7
2	0.0225	0.0430	0.0420	0.0470	0.0338	0.0488	0.0546	0.0563
4	0.0371	0.0450	0.0321	0.0443	0.0270	0.0510	0.0390	0.0540
6	0.0295	0.0170	0.0200	0.0245	0.0360	0.0050	0.0100	0.0105
8	0.0055	-0.0275	-0.0220	-0.0170	-0.0050	-0.0395	-0.0270	-0.0230
10	-0.0062	-0.0342	-0.0287	-0.0237	0.0011	-0.0300	-0.0268	-0.0225
12	0.0142	-0.0112	-0.0212	-0.0121	0.0058	-0.0142	-0.0200	-0.0162
14	0.0013	-0.0317	-0.0275	-0.0185	0.0165	-0.0215	-0.024	-0.0175

Tabla 6.58: *Comparación entre los resultados aplicando la función sigmoidal y PCA en German*

En este caso se empeoran los resultados hasta en 26 ocasiones y en las 30 restantes se obtiene una mejora. A pesar de esto, se sigue produciendo una ligera mejora media de 0,51% (100 iteraciones) y 0,57% (1.000 iteraciones).

6.4.2.2. Análisis de los resultados obtenidos en los experimentos añadiendo 8 atributos adicionales y rotando los datos

En este caso se va a realizar el análisis sobre el conjunto de datos conocido como *German* al que se le han añadido 8 atributos adicionales cuyos valores han sido calculados de forma aleatoria y, posteriormente, se ha llevado a cabo una rotación de los mismos.

Como se muestra en la *Tabla 6.46,* mediante KNN se obtiene un porcentaje de acierto de 71% sobre el conjunto de datos con 32 atributos.

Con la reducción de la dimensionalidad hecha por la transformación lineal se alcanza una tasa de acierto de 73,20% con 4 neuronas, que es superior a la obtenida en los datos originales (71%). Por tanto, se produce una mejora puesto que con un nuevo conjunto de datos con 4 atributos se supera el porcentaje de acierto sobre los datos originales que tienen 32 atributos.

También se mejora el resultado arrojado por KNN sobre los datos originales utilizando la transformación lineal normalizada, logrando con este método un 72,46%. Esta mejora se produce utilizando 10 neuronas.

En el caso de la reducción de la dimensionalidad con la red de neuronas aplicando la función sigmoidal se obtiene una tasa de acierto máxima de 71,71%, utilizando 2 neuronas en la capa oculta. Por tanto, nuevamente se mejora el resultado obtenido en la clasificación de los datos originales (71%) y, además, es importante destacar que esta mejora se consigue con un conjunto de datos con 2 atributos.

Comparación entre los resultados de la transformación lineal y PCA

En la *Tabla 6.59* se muestra una tabla comparativa entre los resultados de la reducción de la dimensionalidad realizada por la transformación lineal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.0563	0.0492	0.0671	0.0829	0.0650	0.0700	0.0690	0.0850
4	0.0365	0.0440	0.0400	0.0295	0.0363	0.0388	0.0250	0.0192
6	0.0150	0.0250	0.0580	0.0520	0.0025	0.0125	0.0490	0.0490
8	0.0065	-0.0010	0.0225	0.0315	0.0240	0.0225	0.0400	0.0385
10	-0.0054	-0.0008	-0.0017	0.0229	-0.0137	-0.0067	-0.0058	0.0163
12	0.0325	-0.0192	0.0042	0.0113	0.0535	0.0025	0.0190	0.0305
14	0.0340	0.0060	0.0090	0.0085	0.0460	0.0215	0.0175	0.0035

Tabla 6.59: *Comparación entre los resultados de la transformación lineal y PCA en German Rotados 8*

En la tabla anterior se observa que se produce un pequeño empeoramiento de los resultados de PCA en algunos casos. A pesar de esto, se consigue una mejora media de 2,56% con la red entrenada con 100 iteraciones y 2,96% con la red entrenada con 1.000 iteraciones. Por otro lado, los resultados máximos de mejora que se logran son 8,29% (100 iteraciones) y 8,50% (1.000 iteraciones).

Los resultados del dominio *German rotados 8* han sido representados en dos gráficas para mejorar la legilibilidad. Por tanto, en la *ilustración 6.9* se pueden apreciar los resultados de la transformación lineal y PCA con *k* igual a 1 y 3, y en la *ilustración 6.10* se muestran los resultados para *k* igual a 5 y 7. En dichas gráficas se observa que de forma general los resultados son mejores con la transformación lineal, pero en ocaciones puntuales PCA consigue alcanzar mejores resultados.

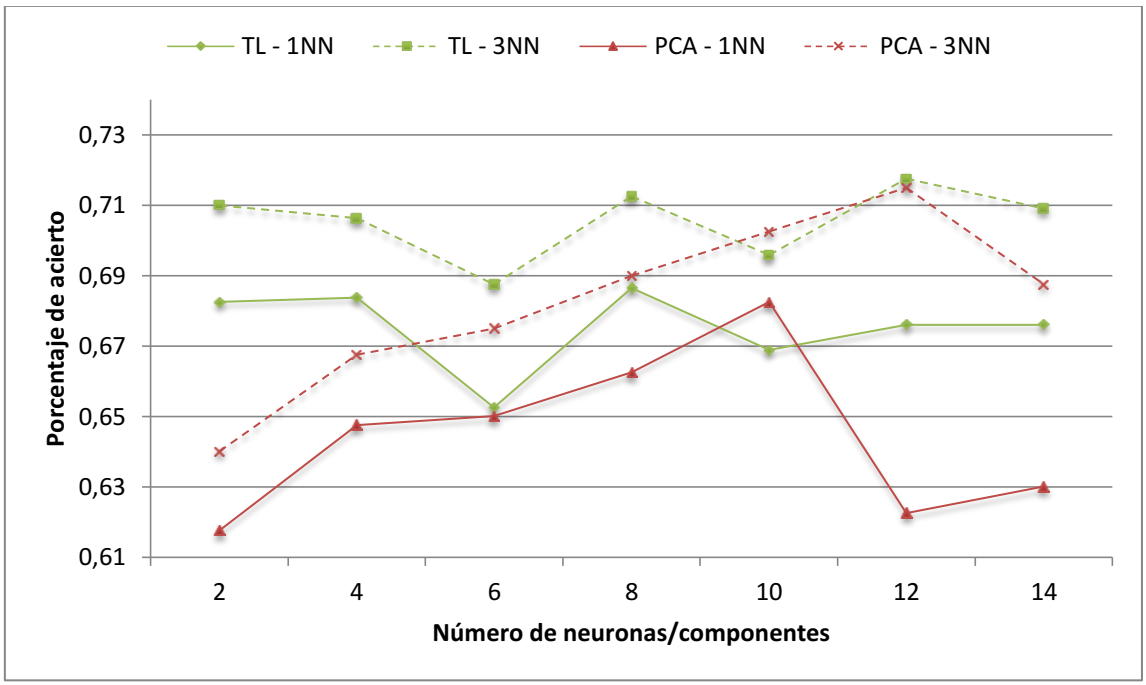

Ilustración 6.9: *Gráfica comparativa de la transformación lineal y PCA en German rotados 8 para k=1 y k=3*

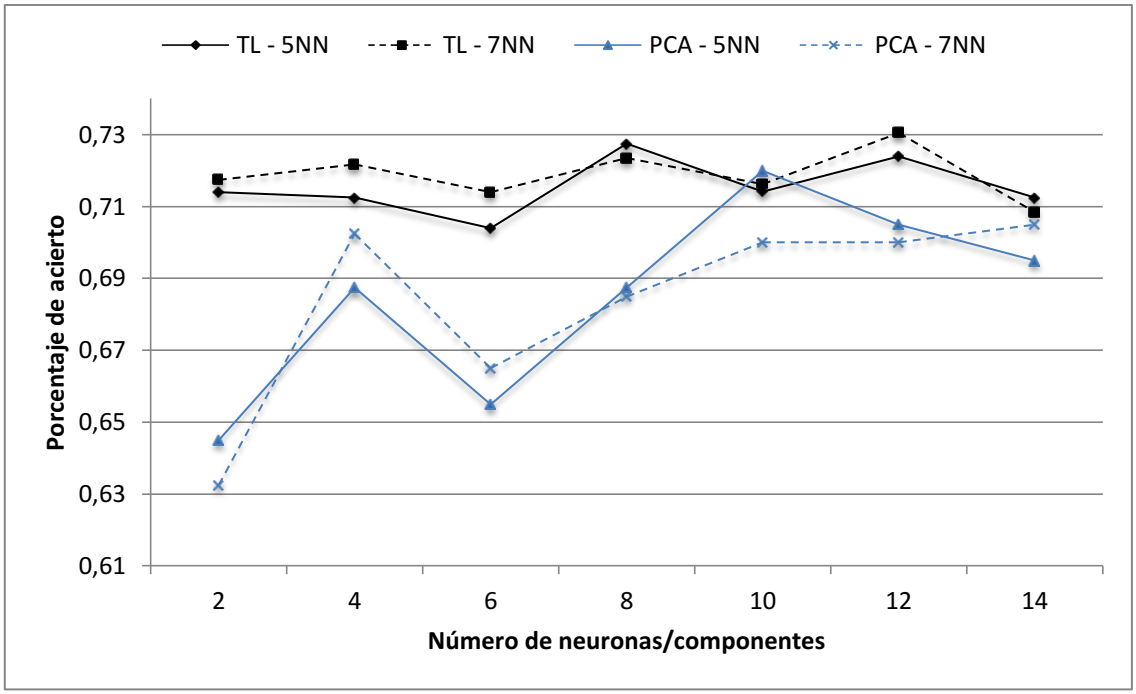

Ilustración 6.10: *Gráfica comparativa de la transformación lineal y PCA en German rotados 8 para k=5 y k=7*

Comparación entre los resultados de la transformación lineal normalizada y PCA

A continuación, se muestra la comparación de la reducción de la dimensionalidad llevada a cabo por la transformación lineal normalizada y PCA. Para obtener los valores de la tabla se ha hecho la resta de los resultados obtenidos con la transformación lineal normalizada y los resultados de PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.0529	0.0600	0.0696	0.0863	0.0620	0.0605	0.0710	0.0860
4	0.0195	0.0440	0.0290	0.0140	0.0217	0.0346	0.0188	0.0092
6	0.0210	0.0200	0.0545	0.0480	0.0090	0.0135	0.0550	0.0485
8	-0.0175	0.0025	0.0150	0.0295	0.0160	0.0045	0.0160	0.0250
10	-0.0062	-0.0025	-0.0029	0.0246	-0.0162	0.0054	-0.0104	0.0108
12	0.0279	-0.0254	-0.0021	0.0117	0.0360	-0.0320	-0.0165	0.0005
14	0.0295	0.0060	0.0085	0.0065	0.0375	0.0105	0.0155	0.0085

Tabla 6.60: *Comparación entre los resultados de la transformación lineal normalizada y PCA en German Rotados 8*

En la *Tabla 6.60* se puede apreciar que en 10 de los 56 casos se empeoran los resultados, pero cabe destacar que esta pérdida en todos los casos es muy pequeña. Adicionalmente, se observa que en el resto de los casos se mejoran los resultados, sin embargo el porcentaje que se mejora no es muy elevado. Por un lado, con 100 iteraciones se obtiene una mejora media en la tasa de acierto del 2,28% y un valor máximo de 8,63%. Por otro lado, con 1.000 iteraciones se obtiene una mejora media de 2,15% y un valor máximo de mejora de 8,60%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la siguiente tabla se muestran los resultados de la comparación de la reducción de la dimensionalidad con la red de neuronas, aplicando la función sigmoidal, y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.0813	0.0733	0.0704	0.0846	0.0630	0.0630	0.0690	0.0840
4	0.0285	0.0250	0.0110	0.0000	0.0350	0.0321	0.0129	0.0004
6	0.0160	0.0010	0.0250	0.0130	0.0150	0.0095	0.0270	0.0230
8	-0.0085	-0.0080	-0.0105	-0.0065	0.0220	0.0050	0.0060	0.0140
10	-0.0096	-0.0092	-0.0233	0.0000	-0.0083	-0.0129	-0.0279	-0.0075
12	0.0483	-0.0312	-0.0200	-0.0158	0.0420	-0.0365	-0.0165	-0.0105
14	0.0570	0.0070	-0.0050	-0.0150	0.0265	-0.0170	-0.0160	-0.0315

Tabla 6.61: *Comparación entre los resultados aplicando la función sigmoidal y PCA en German Rotados8*

En este caso se obtienen peores resultados hasta en 22 ocasiones y, además, en 2 casos más se registra un porcentaje idéntico, es decir, existen 2 valores en la tabla que son 0. Sin embargo, la media de los valores de la tabla arroja que se sigue produciendo mejora con respecto a PCA con la red entrenada con 100 y 1.000 iteraciones, alcanzando un porcentaje de 1,35% y 2,14%, respectivamente. Además, se obtienen valores máximos de mejora casi idénticos.

6.4.2.3. Análisis de los resultados obtenidos en los experimentos añadiendo 24 atributos adicionales y rotando los datos

Como se observa en la *Tabla 6.51*, el mejor porcentaje de acierto obtenido por KNN al realizar la tarea de clasificación sobre los datos originales es de 69,25%. En este caso el conjunto de datos original está formado por 48 atributos.

Por otro lado, se observa que un 71,75% (18 neuronas en la capa oculta) es la mejor tasa de acierto conseguida por KNN sobre la reducción de la dimensionalidad realizada por la transformación lineal. Por tanto, se puede ver que se mejora ligeramente el resultado de clasificación obtenido sobre el conjunto datos original (69,25%). Cabe destacar que esta mejora se produce con el conjunto de datos transformado con un menor número de dimensiones, es decir, se alcanza un mejor resultado con el conjunto de datos con 18 atributos frente a los 48 atributos del conjunto de datos original.

También es rentable la reducción de la dimensionalidad con la transformación lineal normalizada, ya que se ofrece un valor máximo de 72,29% con 24 neuronas en la capa oculta y se mejora el porcentaje de acierto alcanzado sobre el conjunto de datos original (69,25%).

En cuanto a la reducción de la dimensionalidad hecha por la red de neuronas, aplicando la función sigmoidal, se observa que se logra un máximo de 70,68% con 20 neuronas en la capa oculta. En este caso también se consigue superar el resultado obtenido en la clasificación de los datos originales (69,25%), sin embargo la mejora que se produce no es tan amplia como en los dos casos anteriores.

En la *Tabla 6.55* se comprueba que el mejor resultado conseguido por PCA es 71,50%. Comparando este valor con la reducción de la dimensionalidad realizada por la red de neuronas se observa que únicamente la transformación lineal normalizada consigue superar el valor obtenido por PCA, pero cabe destacar que la diferencia de PCA con los otros dos casos (transformación lineal y sigmoidal) no es significativa ya que en ambos casos es menor a un 1%.

Comparación entre los resultados de la transformación lineal y PCA

En la siguiente tabla se muestra una comparativa entre los valores registrados por la reducción de la dimensionalidad llevada a cabo por la transformación lineal y PCA. La tabla ha sido calculada obteniendo la resta entre los resultados de la transformación lineal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.0250	0.0240	0.0430	0.0390	0.0289	0.0189	0.0511	0.0379
6	0.0020	0.0135	0.0015	-0.0335	-0.0050	0.0045	0.0070	-0.0160
10	0.0210	0.0360	0.0450	0.0280	0.0255	0.0320	0.0420	0.0250
14	0.0580	0.0360	0.0265	0.0125	0.0516	0.0313	0.0222	0.0044
18	0.0390	0.0590	0.0425	0.0475	0.0242	0.0433	0.0358	0.0354
20	0.0536	0.0193	0.0279	0.0339	0.0429	0.0171	0.0325	0.0296
24	0.0654	0.0400	0.0207	0.0325	0.0640	0.0460	0.0215	0.0265

Tabla 6.62: *Comparación entre los resultados de la transformación lineal y PCA en German Rotados 24*

En la tabla se observa como existen tres valores negativos, esto quiere decir que en estos casos se obtiene un peor resultado con la transformación lineal. Sin embargo, cabe destacar que estos valores negativos están muy próximos a cero, lo que significa que la pérdida es prácticamente nula en ambos casos. Adicionalmente, se observa que en el resto de situaciones (53) los resultados son favorables, llegando a obtener con 1.000 iteraciones un 6,40% y con 100 iteraciones un 6,54% de beneficio máximo. Por otro lado, calculando la media de los valores de la tabla con 100 iteraciones se alcanza un 3,07% y con 1.000 iteraciones 2,79%.

En las siguientes gráficas se muestran los resultados obtenidos por la transformación lineal y PCA en el conjunto de datos *German rotados 24*. En este caso se han utilizado dos gráficas para representar los datos y tener una mejor visualización de los mismos. Al igual que en el caso anterior, se observa que los resultados de la transformación lineal, de forma genérica, son superiores.

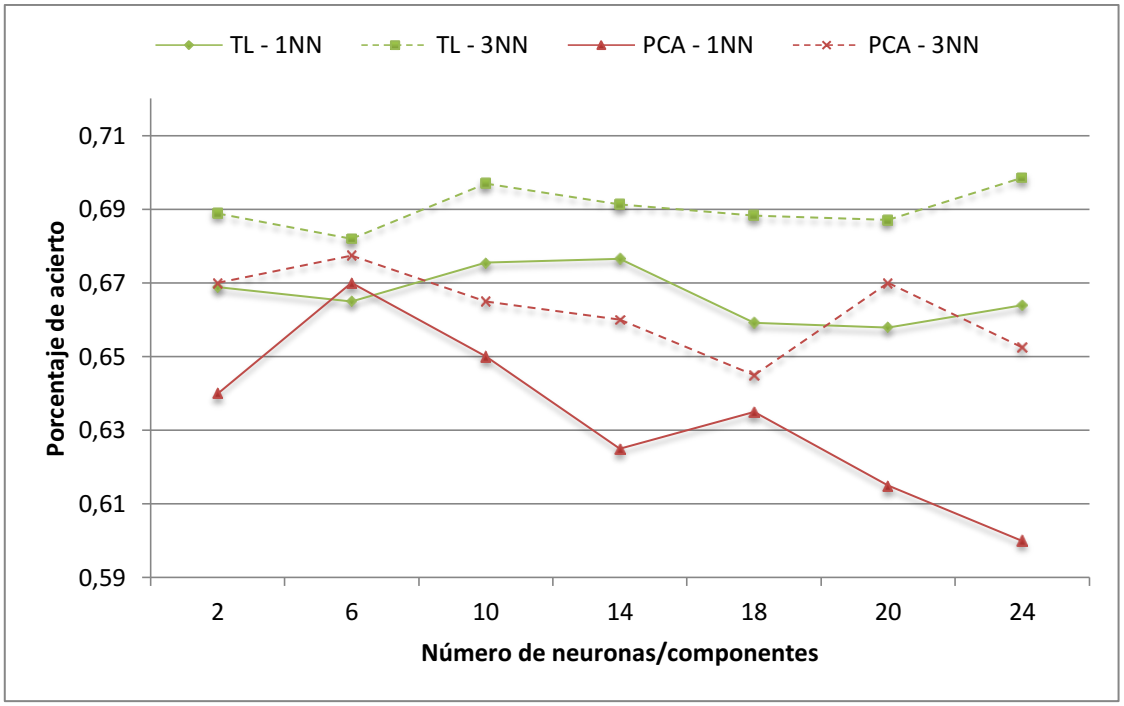

Ilustración 6.11: *Gráfica comparativa de la transformación lineal y PCA en German rotados 24 para k=1 y k=3*

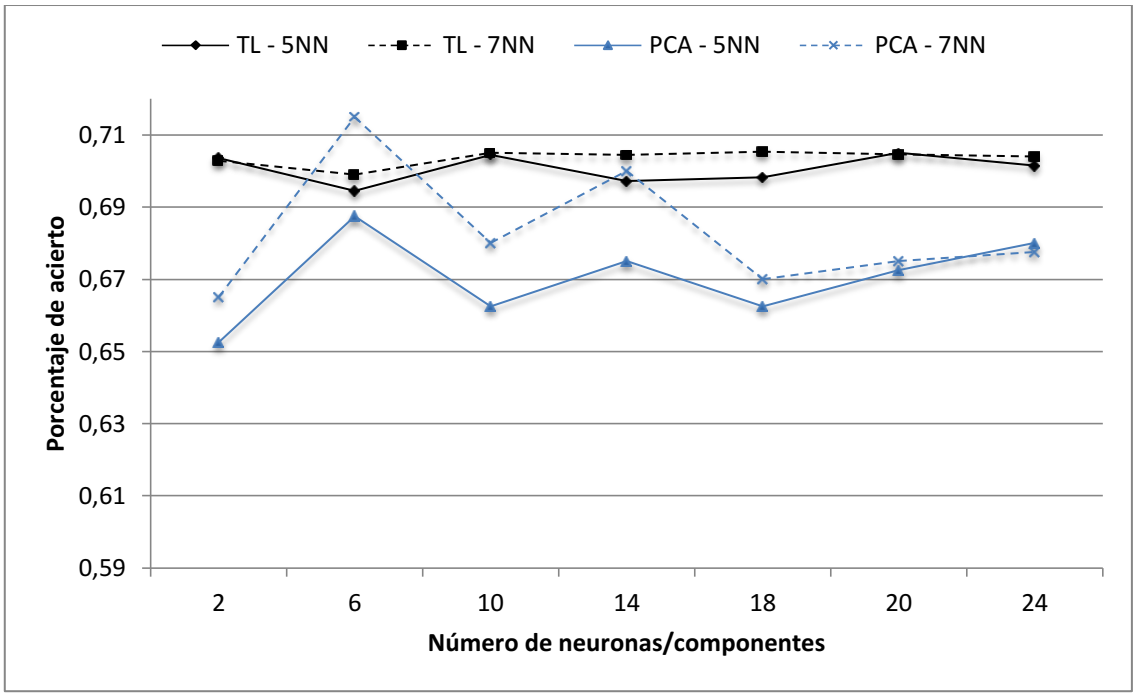

Ilustración 6.12: *Gráfica comparativa de la transformación lineal y PCA en German rotados 24 para k=5 y k=7*

Comparación entre los resultados de la transformación lineal normalizada y PCA

A continuación, se muestra la tabla comparativa entre la transformación lineal normalizada y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.0075	0.0170	0.0440	0.0330	0.0307	0.0129	0.0386	0.0336
6	-0.0130	0.0050	-0.0130	-0.0305	0.0085	0.0275	0.0210	-0.0090
10	0.0215	0.0380	0.0510	0.0350	0.0305	0.0340	0.0485	0.0385
14	0.0460	0.0435	0.0370	0.0110	0.0491	0.0341	0.0278	0.0072
18	0.0345	0.0485	0.0470	0.0470	0.0333	0.0554	0.0388	0.0492
20	0.0529	0.0336	0.0379	0.0318	0.0446	0.0158	0.0300	0.0333
24	0.0543	0.0411	0.0339	0.0454	0.0650	0.0425	0.0370	0.0435

Tabla 6.63: *Comparación entre los resultados de la transformación lineal normalizada y PCA en German Rotados 24*

En la tabla se aprecian cuatro valores negativos, mientras que en el resto de ocasiones (52) se obtiene un beneficio al realizar la reducción de la dimensionalidad utilizando la transformación lineal normalizada. Por tanto, entrenando la red de neuronas con 1.000 iteraciones se observa una mejora media en el porcentaje de acierto de 3,29%, llegando a obtener un máximo valor de beneficio de 6,50%. Con la red entrenada con 100 iteraciones se consiguen unos resultados ligeramente inferiores puesto que la mejora media obtenida es de 3% y el máximo alcanzado es 5,43%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.64* se muestra una comparación entre la reducción de la dimensionalidad producida por la red de neuronas aplicando la función sigmoidal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
2	0.0125	0.0070	0.0495	0.0395	0.0307	0.0286	0.0464	0.0404
6	0.0005	0.0130	0.0085	-0.0185	0.0160	0.0150	0.0105	-0.0120
10	0.0370	0.0370	0.0400	0.0230	0.0345	0.0375	0.0400	0.0235
14	0.0360	0.0080	0.0040	-0.0200	0.0597	0.0359	0.0291	0.0019
18	0.0445	0.0575	0.0415	0.0345	0.0438	0.0438	0.0329	0.0221
20	0.0700	0.0329	0.0343	0.0311	0.0613	0.0196	0.0204	0.0188
24	0.0704	0.0300	0.0057	0.0143	0.0780	0.0365	0.0125	0.0150

Tabla 6.64: *Comparación entre los resultados aplicando la función sigmoidal y PCA en German Rotados24*

Nuevamente, en la tabla se registran tres valores negativos que son muy próximos a cero, lo que indica que en estos casos la pérdida es mínima y no es significativa. En el resto de situaciones se consiguen valores positivos. Cabe destacar que con 1.000 iteraciones el mayor valor que se alcanza es de 7,80% y la tasa media de acierto alcanzada con la red de neuronas es superior en un 3%. Por otro lado, entrenando la red con 100 iteraciones se obtiene una mejora media de 2,66% y un máximo de 7,04%.

6.5. Dominio *Splice*

El dominio conocido con el nombre de *Splice* contiene un conjunto de datos formados por 60 atributos con información relativa a una secuencia de ADN. Además, cada instancia contiene su clase correspondiente que puede tomar dos posibles valores: si la instancia es reconocida como límite exón/intrón o, por el contrario, si es reconocida como intrón/exón. Cabe destacar que el conjunto de datos contiene un total de 1.000 instancias.

6.5.1. Experimentos realizados sobre el dominio *Splice*

6.5.1.1. Experimentos sobre los datos originales

En primer lugar, se muestran los resultados de los experimentos ejecutados sobre el conjunto original de datos del dominio *Splice*. Por tanto, el conjunto de datos utilizados en este caso contiene 60 atributos y la clase.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.7050
3	0.7200
5	0.7200
7	0.7275

Tabla 6.65: *Resultados de KNN sobre los datos originales en Splice*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

A continuación, en la *Tabla 6.66,* se puede apreciar el resultado que proporciona KNN sobre la transformación lineal que lleva a cabo la capa oculta de la red.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
4	0.7525	0.7715	0.7700	0.7790	0.7685	0.7720	0.7740	0.7730
8	0.8013	0.8058	0.8096	0.8188	0.7815	0.7790	0.7770	0.7825
12	0.8125	0.8275	0.8229	0.8246	0.8197	0.8247	0.8322	0.8341
16	0.8117	0.8225	0.8213	0.8163	0.8138	0.8219	0.8234	0.8244
20	0.8168	0.8248	0.8295	0.8318	0.8119	0.8292	0.8306	0.8328
24	0.8169	0.8281	0.8266	0.8288	0.8161	0.8204	0.8250	0.8218
28	0.8006	0.8121	0.8123	0.8158	0.7984	0.8082	0.8084	0.8102

Tabla 6.66: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Splice*

El mayor porcentaje de acierto conseguido es de 83,18% en el caso de la transformación lineal con la red entrenada con 100 iteraciones, mientras que con la red entrenada con 1.000 iteraciones el mejor resultado que se registra es de 83,41%. Teniendo en cuenta esto, el mejor resultado obtenido con la transformación lineal es de 83,41% con 12 neuronas en la capa oculta.

En la *Tabla 6.67* se pueden observar los resultados obtenidos por KNN sobre la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
4	0.7400	0.7570	0.7540	0.7625	0.7510	0.7630	0.7610	0.7625
8	0.7838	0.7946	0.8038	0.8013	0.7600	0.7760	0.7810	0.7815
12	0.7983	0.8200	0.8254	0.8183	0.7997	0.8153	0.8194	0.8197
16	0.7900	0.8021	0.8100	0.8113	0.7931	0.8088	0.8169	0.8112
20	0.8040	0.8178	0.8253	0.8288	0.8081	0.8189	0.8164	0.8222
24	0.7934	0.8163	0.8178	0.8197	0.7939	0.8025	0.8036	0.8100
28	0.7765	0.7910	0.7952	0.7950	0.7904	0.8005	0.8002	0.8016

Tabla 6.67: *Resultados de KNN sobre la transformación lineal normalizada en Splice*

Comparando los resultados de la *Tabla 6.67*, se observa que el mejor resultado ofrecido por la transformación lineal normalizada es de 82,88% con 20 neuronas en la capa oculta y habiendo realizado el entrenamiento de la red de neuronas con 100 iteraciones.

En la *Tabla 6.68* se muestran los resultados arrojados por KNN en la clasificación de la reducción de la dimensionalidad producida por la red de neuronas, aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
4	0.7475	0.7560	0.7550	0.7555	0.7450	0.7550	0.7555	0.7560
8	0.7896	0.7917	0.7963	0.7996	0.7750	0.7680	0.7690	0.7680
12	0.8021	0.8096	0.8108	0.8092	0.8113	0.8072	0.8078	0.8103
16	0.8104	0.8083	0.8083	0.8063	0.8094	0.8078	0.8066	0.8044
20	0.7970	0.8005	0.8015	0.8030	0.8131	0.8128	0.8111	0.8122
24	0.8228	0.8200	0.8191	0.8200	0.8221	0.8204	0.8168	0.8161
28	0.8127	0.8142	0.8115	0.8113	0.8091	0.8100	0.8071	0.8039

Tabla 6.68: *Resultados de KNN aplicando la función sigmoidal en Splice*

Analizando los valores de la *Tabla 6.68* se observa que con 100 iteraciones el mejor porcentaje de acierto es de 82,28%, mientras que con 1.000 iteraciones el mejor valor obtenido es de 82,21%. Por tanto, en este caso el mejor resultado que se alcanza con la reducción de la dimensionalidad hecha por la red de neuronas aplicando la función sigmoidal es de 82,28%. Este valor se registra con 24 neuronas en la capa oculta.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
4	0.6750	0.7325	0.7600	0.7400
8	0.7550	0.7375	0.7475	0.7525
12	0.7150	0.7400	0.7650	0.7825
16	0.7575	0.7875	0.7825	0.7850
20	0.7225	0.7700	0.7750	0.7925
24	0.7350	0.7675	0.7925	0.7725
28	0.7175	0.7450	0.7425	0.7650

Tabla 6.69: *Resultados de KNN sobre los datos transformados por PCA en Splice*

6.5.1.2. Experimentos añadiendo 8 atributos adicionales y rotando los datos

En este apartado se muestran los resultados obtenidos añadiendo 8 atributos adicionales, calculados de forma aleatoria en el rango [0,1], al conjunto de datos original del dominio *Splice* y, posteriormente, realizando una rotación de los mismos. Por tanto, los datos que han sido utilizados en este caso contienen un total de 68 atributos y la clase correspondiente.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.6842
3	0.6642
5	0.6892
7	0.6742

Tabla 6.70: *Resultados de KNN sobre los datos originales en Splice Rotados 8*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.71* se muestra la tasa media de acierto obtenida por KNN sobre la reducción de la dimensionalidad llevada a cabo por la transformación lineal que produce la capa oculta de la red de neuronas.

	100 iteraciones				1.000 iteraciones			
Neuronas \ Vecinos	1	3	5	7	1	3	5	7
5	0.7494	0.7664	0.7734	0.7769	0.7531	0.7690	0.7723	0.7719
10	0.7825	0.7945	0.7980	0.7965	0.7880	0.8015	0.8000	0.7980
15	0.8067	0.8204	0.8279	0.8266	0.7945	0.7985	0.7985	0.8015
20	0.8028	0.8133	0.8143	0.8103	0.8049	0.8188	0.8178	0.8181
25	0.7989	0.8067	0.8029	0.8073	0.8108	0.8183	0.8191	0.8191
30	0.8095	0.8033	0.8112	0.8099	0.8088	0.8084	0.8170	0.8213
35	0.8056	0.8129	0.8159	0.8157	0.8153	0.8163	0.8167	0.8210

Tabla 6.71: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Splice Rotados 8*

En la *Tabla 6.71* se observa que el mejor resultado conseguido por la transformación lineal, entrenando la red con 100 iteraciones, es de 82,79%. Por otro lado, se observa que este porcentaje no es superado con la red entrenada con 1.000 iteraciones. Teniendo en cuenta estos datos, el mejor resultado alcanzado en el caso de la transformación lineal es de 82,79% con 15 neuronas en la capa oculta.

En la siguiente tabla se muestra la tasa de acierto obtenida por KNN sobre la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
5	0.7529	0.7754	0.7805	0.7799	0.7548	0.7690	0.7749	0.7740
10	0.7820	0.7985	0.8025	0.8000	0.7724	0.7779	0.7850	0.7850
15	0.7857	0.8074	0.8129	0.8158	0.7764	0.7945	0.7970	0.7975
20	0.7895	0.7945	0.7990	0.8000	0.7816	0.7995	0.8077	0.8099
25	0.7804	0.7951	0.7982	0.7967	0.7907	0.7991	0.8116	0.8083
30	0.7765	0.7970	0.7966	0.8070	0.7909	0.8020	0.8031	0.8020
35	0.7817	0.7909	0.7995	0.8006	0.7898	0.8034	0.8095	0.8081

Tabla 6.72: *Resultados de KNN sobre la transformación lineal normalizada en Splice Rotados 8*

En el caso de la transformación lineal normalizada, comparando los resultados de la red entrenada con 100 y 1.000 iteraciones, se observa que la mejor tasa de acierto registrada es de 81,58% con 15 neuronas en la capa oculta.

En la *Tabla 6.73* se muestran los resultados logrados por KNN sobre la reducción de la dimensionalidad realizada por la red de neuronas aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
5	0.7604	0.7639	0.7669	0.7684	0.7540	0.7590	0.7632	0.7627
10	0.7830	0.7825	0.7860	0.7830	0.7704	0.7714	0.7729	0.7734
15	0.7982	0.8053	0.8079	0.8095	0.7960	0.7925	0.7915	0.7915
20	0.8083	0.8105	0.8070	0.8045	0.8045	0.8059	0.8074	0.8063
25	0.7961	0.7973	0.7973	0.7957	0.8049	0.8087	0.8137	0.8104
30	0.7949	0.8008	0.7982	0.8016	0.8178	0.8113	0.8153	0.8160
35	0.8079	0.8162	0.8168	0.8179	0.8195	0.8238	0.8192	0.8178

Tabla 6.73: *Resultados de KNN aplicando la función sigmoidal en Splice Rotados 8*

Como se muestra en la *Tabla 6.73*, el mejor porcentaje obtenido aplicando la función sigmoidal y entrenando la red con 100 iteraciones es de 81,79%. Por otro lado, si observamos los resultados con la red entrenada con 1.000 iteraciones, se aprecia que el mayor valor registrado es de 82,38%. Por tanto, en este caso el mejor resultado obtenido con 35 neuronas es de 82,38%.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
5	0.6917	0.6792	0.6817	0.6817
10	0.7018	0.6892	0.7168	0.7268
15	0.7018	0.7293	0.7293	0.7494
20	0.7093	0.7444	0.7494	0.7368
25	0.6767	0.7243	0.7343	0.7243
30	0.6867	0.7193	0.7218	0.7268
35	0.7068	0.6817	0.7068	0.7043

Tabla 6.74: *Resultados de KNN sobre los datos transformados por PCA en Splice Rotados 8*

6.5.1.3. Experimentos añadiendo 20 atributos adicionales y rotando los datos

En este apartado se incluyen los resultados obtenidos añadiendo 20 atributos adicionales aleatorios al conjunto de datos original, manteniendo los 60 atributos originales y su clase correspondiente. Teniendo en cuenta esto, el conjunto de datos utilizados en este caso para realizar las pruebas contienen 80 atributos en total y la clase.

Tasa media de acierto obtenida utilizando KNN sobre los datos originales

Número de vecinos (parámetro *k*)	Tasa media de acierto
1	0.7043
3	0.6992
5	0.7118
7	0.7193

Tabla 6.75: *Resultados de KNN sobre los datos originales en Splice Rotados 20*

Tasa media de acierto obtenida utilizando KNN sobre las activaciones de las neuronas de la capa oculta

En la *Tabla 6.76* se pueden apreciar la tasa media de acierto que proporciona KNN sobre la transformación lineal que produce la capa oculta de la red.

	100 iteraciones				1.000 iteraciones			
Neuronas \ Vecinos	1	3	5	7	1	3	5	7
5	0.7484	0.7604	0.7629	0.7659	0.7782	0.7820	0.7845	0.7853
10	0.7941	0.7995	0.8024	0.8028	0.7880	0.7915	0.7905	0.7945
15	0.8013	0.8084	0.8124	0.8131	0.7855	0.7910	0.7910	0.7920
20	0.8099	0.8170	0.8199	0.8217	0.8102	0.8264	0.8238	0.8271
25	0.8158	0.8253	0.8286	0.8268	0.8102	0.8130	0.8195	0.8195
30	0.8033	0.8133	0.8170	0.8162	0.8099	0.8179	0.8204	0.8124
35	0.8170	0.8229	0.8175	0.8216	0.8183	0.8216	0.8208	0.8237

Tabla 6.76: *Resultados de KNN sobre la transformación lineal que realiza la capa oculta de la red en Splice Rotados 20*

Como se puede observar en la *Tabla 6.76,* el mejor resultado que se registra con la transformación lineal y la red entrenada con 100 iteraciones es de 82,86%, mientras que con 1.000 iteraciones el mejor resultado logrado es 82,71%. Por tanto, el mejor resultado de reducción de dimensionalidad con la transformación lineal es 82,86% que se consigue con 25 neuronas en la capa oculta y 7 vecinos.

A continuación, en la *Tabla 6.77*, se muestra la tasa de acierto obtenida en la tarea de clasificación por KNN sobre la transformación lineal normalizada.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
5	0.7504	0.7609	0.7744	0.7749	0.7556	0.7640	0.7740	0.7774
10	0.7627	0.7949	0.8012	0.7945	0.7825	0.7930	0.7930	0.7975
15	0.7995	0.8081	0.8102	0.8127	0.7684	0.7845	0.7940	0.7935
20	0.7984	0.8131	0.8102	0.8138	0.8056	0.8099	0.8210	0.8206
25	0.7870	0.8033	0.8058	0.8042	0.7854	0.7951	0.7945	0.8017
30	0.7891	0.8045	0.8108	0.8091	0.7865	0.8012	0.8104	0.8091
35	0.8024	0.8129	0.8187	0.8208	0.7962	0.8028	0.8041	0.8120

Tabla 6.77: *Resultados de KNN sobre la transformación lineal normalizada en Splice Rotados 20*

El mejor resultado conseguido con la transformación lineal normalizada y la red entrenada con 100 iteraciones es 82,08%. Sin embargo, entrenando la red con 1.000 iteraciones se obtiene 82,10% que es ligeramente superior al valor anterior. Por ello, el mejor porcentaje de acierto alcanzado es 82,10% que corresponde con 20 neuronas en la capa oculta y 5 vecinos.

En la *Tabla 6.78* se muestra la tasa de acierto obtenida por KNN sobre la reducción de la dimensionalidad realizada por la red de neuronas aplicando la función sigmoidal.

	100 iteraciones				1.000 iteraciones			
Vecinos / Neuronas	1	3	5	7	1	3	5	7
5	0.7469	0.7549	0.7569	0.7554	0.7673	0.7657	0.7657	0.7673
10	0.7703	0.7761	0.7769	0.7786	0.7694	0.7724	0.7754	0.7754
15	0.7938	0.8013	0.8020	0.7984	0.7764	0.7744	0.7724	0.7759
20	0.7995	0.8049	0.8088	0.8084	0.7981	0.7999	0.8013	0.8006
25	0.8130	0.8164	0.8164	0.8189	0.8064	0.8076	0.8073	0.8070
30	0.8062	0.8079	0.8108	0.8091	0.8087	0.8079	0.8079	0.8099
35	0.8166	0.8200	0.8170	0.8158	0.8292	0.8241	0.8246	0.8200

Tabla 6.78: *Resultados de KNN aplicando la función sigmoidal en Splice Rotados 20*

En el caso de la reducción de la dimensionalidad aplicando la función sigmoidal se observa que haciendo el entrenamiento con 100 iteraciones el máximo porcentaje de acierto obtenido es de 82%, mientras que con 1.000 iteraciones el resultado es de 82,92%. Teniendo en cuenta estos valores, el mejor resultado alcanzado es de 82,92% que corresponde con 35 neuronas en la capa oculta y 1 vecino.

Tasa media de acierto obtenida utilizando KNN sobre los datos transformados por PCA

Componentes \ Vecinos	1	3	5	7
5	0.6667	0.7043	0.7293	0.7444
10	0.7018	0.7068	0.7068	0.7143
15	0.7168	0.7168	0.7043	0.6862
20	0.6867	0.7218	0.7043	0.7193
25	0.6591	0.6967	0.7218	0.7444
30	0.6867	0.7018	0.7093	0.7243
35	0.6717	0.6792	0.7143	0.7068

Tabla 6.79: *Resultados de KNN sobre los datos transformados por PCA*

6.5.2. Análisis de los resultados obtenidos en el dominio *Splice*

6.5.2.1. Análisis de los resultados obtenidos en los experimentos sobre los datos originales

El mejor porcentaje de clasificación conseguido sobre el conjunto de datos llamado *Splice* ha sido 72,75%.

Por otro lado, el mejor resultado ofrecido con la transformación lineal es 83,41% con 12 neuronas en la capa oculta. Como se puede comprobar, el resultado alcanzado habiendo realizado la reducción de la dimensionalidad es mejor que el que se obtiene sobre los datos originales (72,75%). Además, cabe destacar que esta mejora en la tasa de acierto es lograda con un conjunto de datos con 12 atributos frente a los 60 atributos que tiene el conjunto de datos *Splice*.

Con la transformación lineal normalizada el mejor resultado de clasificación logrado es de 82,88% con 20 neuronas en la capa oculta. Este resultado también mejora de forma amplia la tasa de acierto obtenida sobre los datos originales.

En cuanto a la reducción de la dimensionalidad llevada a cabo aplicando la función sigmoidal, el mejor valor de clasificación que se registra es 82,28% con 24 neuronas en la capa oculta. Por ello, utilizando este método de reducción de dimensionalidad también se logran beneficios al superar los resultados de clasificación sobre los datos originales con un nuevo conjunto de datos que tiene un número menor de dimensiones. Sin embargo, en este caso se observa que la transformación lineal proporciona resultados superiores que aplicando la función sigmoidal, puesto que se obtiene un resultado mejor en la clasificación y el conjunto de datos generado contiene un número menor de dimensiones.

En la *Tabla 6.69* se observa que el mejor resultado logrado por PCA corresponde con 79,25% utilizando 20 componentes. Este porcentaje, aunque supera el resultado de clasificación sobre los datos originales, no consigue mejorar los resultados que proporciona la transformación lineal (83,41%), transformación lineal normalizada (82,88%) y aplicando la función sigmoidal (82,28%).

A continuación, se muestra una comparación más detallada de los resultados obtenidos con la red de neuronas y PCA.

Comparación entre los resultados de la transformación lineal y PCA

En la siguiente tabla se muestra la comparativa entre la reducción de la dimensionalidad realizada por la transformación lineal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
4	0.0775	0.0390	0.0100	0.0390	0.0935	0.0395	0.0140	0.0330
8	0.0463	0.0683	0.0621	0.0663	0.0265	0.0415	0.0295	0.0300
12	0.0975	0.0875	0.0579	0.0421	0.1047	0.0847	0.0672	0.0516
16	0.0542	0.0350	0.0388	0.0313	0.0563	0.0344	0.0409	0.0394
20	0.0943	0.0548	0.0545	0.0393	0.0894	0.0592	0.0556	0.0403
24	0.0819	0.0606	0.0341	0.0563	0.0811	0.0529	0.0325	0.0493
28	0.0831	0.0671	0.0698	0.0508	0.0809	0.0632	0.0659	0.0452

Tabla 6.80: *Comparación entre los resultados de la transformación lineal y PCA en Splice*

Como se puede observar en la tabla, se mejoran todos los resultados de PCA al obtener valores positivos en todas las situaciones. Cabe destacar que con 100 iteraciones el valor máximo alcanzado es 9,75% y una mejora media de 5,71%. Por otro lado, entrenando la red con 1.000 iteraciones se logra un máximo valor de mejora de 10,47% y una mejora media de 5,37%.

En la *ilustración 6.13* se muestran los resultados de la transformación lineal y PCA en el dominio *Splice*. En dicha imagen se observa que todos los resultados de la transformación lineal son superiores a PCA.

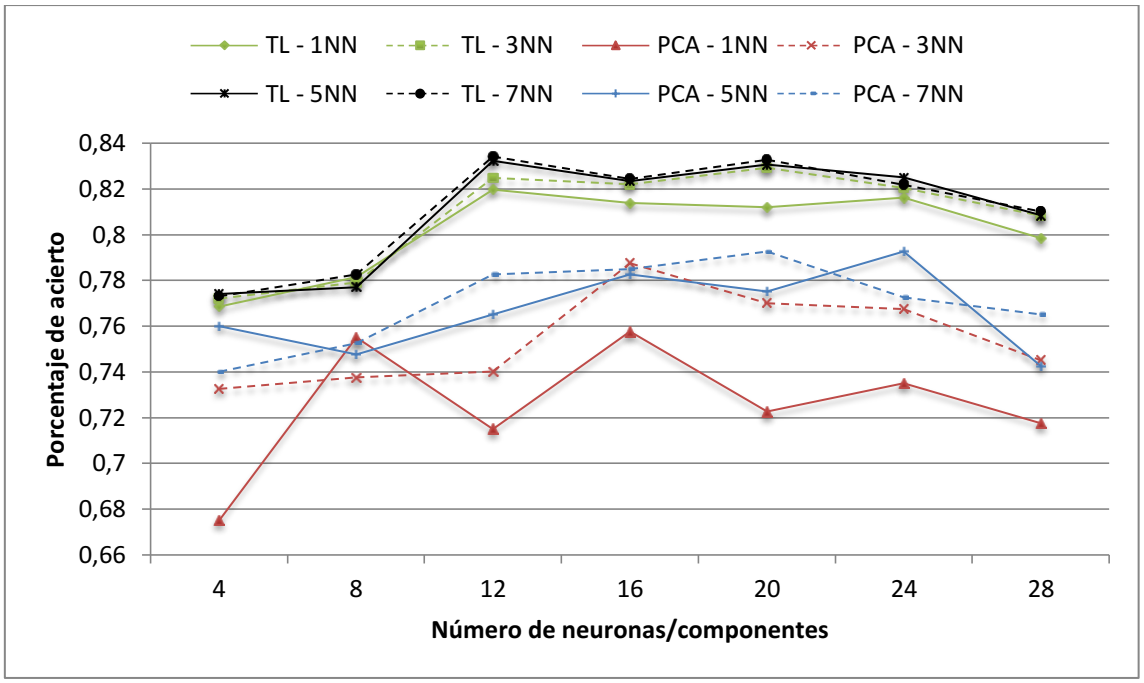

Ilustración 6.13: *Gráfica comparativa de la transformación lineal y PCA en Splice*

Comparación entre los resultados de la transformación lineal normalizada y PCA

En la *Tabla 6.81* se muestra la comparación calculada entre los resultados de la transformación lineal normalizada y PCA. Esta comparación ha sido hecha calculando

la resta de los resultados obtenidos con la transformación lineal normalizada (con la red entrenada con 100 y 1.000 iteraciones por separado) y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
4	0.0650	0.0245	-0.0060	0.0225	0.0760	0.0305	0.0010	0.0225
8	0.0288	0.0571	0.0563	0.0488	0.0050	0.0385	0.0335	0.0290
12	0.0833	0.0800	0.0604	0.0358	0.0847	0.0753	0.0544	0.0372
16	0.0325	0.0146	0.0275	0.0263	0.0356	0.0213	0.0344	0.0262
20	0.0815	0.0478	0.0503	0.0363	0.0856	0.0489	0.0414	0.0297
24	0.0584	0.0488	0.0253	0.0472	0.0589	0.0350	0.0111	0.0375
28	0.0590	0.046	0.0527	0.0300	0.0729	0.0555	0.0577	0.0366

Tabla 6.81: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Splice*

En la tabla anterior se observa un único caso en el que se empeora el resultado de PCA, pero la pérdida que se produce es muy pequeña (0,6%). En el resto de situaciones el resultado obtenido en la comparación es positivo. Por tanto, entrenando la red con 1.000 iteraciones la mejora media obtenida es de 4,20% y se alcanza un valor máximo de mejora de 8,56%. Un valor prácticamente idéntico es obtenido con la red entrenada con 100 iteraciones al lograr una mejora media de 4,43% y un máximo de 8,33%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.82* se muestra la comparación llevada a cabo entre la tasa de acierto media obtenida en la reducción de la dimensionalidad producida por la red de neuronas, aplicando la función sigmoidal, y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
4	0.0725	0.0235	-0.0050	0.0155	0.0700	0.0225	-0.0045	0.0160
8	0.0346	0.0542	0.0488	0.0471	0.0200	0.0305	0.0215	0.0155
12	0.0871	0.0696	0.0458	0.0267	0.0963	0.0672	0.0428	0.0278
16	0.0529	0.0208	0.0258	0.0213	0.0519	0.0203	0.0241	0.0194
20	0.0745	0.0305	0.0265	0.0105	0.0906	0.0428	0.0361	0.0197
24	0.0878	0.0525	0.0266	0.0475	0.0871	0.0529	0.0243	0.0436
28	0.0952	0.0692	0.0690	0.0463	0.0916	0.0650	0.0646	0.0389

Tabla 6.82: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Splice*

Observando la tabla anterior se detectan dos situaciones en las que se registran resultados negativos que están muy próximos a cero, es decir, la pérdida en estos casos es prácticamente inexistente. Por un lado, entrenando la red con 100 iteraciones la mejora media obtenida es de 4,56% y se alcanza un máximo de 9,52% y, por otro lado, con 1.000 iteraciones la mejora media es de 4,28% con un valor máximo de 9,63%.

6.5.2.2. Análisis de los resultados obtenidos en los experimentos añadiendo 8 atributos adicionales y rotando los datos

Como se observa en la *Tabla 6.70*, el máximo porcentaje de acierto obtenido en la clasificación de los datos originales es de 68,92%.

Por otro lado, en la *Tabla 6.71* se observa que realizando la transformación lineal se obtiene un resultado máximo de 82,79% con 15 neuronas en la capa oculta. Nótese que el resultado de clasificación sobre los datos originales (68,92%) es superado ampliamente con la transformación lineal y, por tanto, en este caso se produce un gran beneficio realizando la reducción de la dimensionalidad, ya que con un conjunto de datos con 15 atributos se supera el resultado de los datos originales con 68 atributos.

También se produce un gran beneficio con la transformación lineal normalizada porque se obtiene un máximo en la clasificación de 81,58% con 15 neuronas en la capa oculta.

Con la reducción de la dimensionalidad realizada aplicando la función sigmoidal se alcanza un máximo de 82,38% con 35 neuronas en la capa oculta. Con esta aproximación se mejoran también de forma amplia los resultados sobre el conjunto de datos original (68,92%). Sin embargo, el conjunto de datos con dimensiones reducidas tendría 35 atributos que es una cantidad muy superior a los 15 atributos del conjunto de datos generado con la transformación lineal o la transformación lineal normalizada.

Cabe destacar que con PCA se obtiene una tasa de acierto máxima de 74,94% con 15 componentes. Este resultado de clasificación es peor que todos los logrados con la red de neuronas.

En las siguientes tablas se muestra una comparación más amplia de la transformación lineal, la transformación lineal normalizada y aplicando la función sigmoidal con PCA. La comparación es realizada con 100 y 1.000 iteraciones por separado.

Comparación entre los resultados de la transformación lineal y PCA

En la siguiente tabla se muestra una comparación entre todos los resultados obtenidos con la transformación lineal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
5	0.0577	0.0872	0.0917	0.0952	0.0614	0.0898	0.0906	0.0902
10	0.0807	0.1053	0.0812	0.0697	0.0862	0.1123	0.0832	0.0712
15	0.1049	0.0911	0.0986	0.0772	0.0927	0.0692	0.0692	0.0521
20	0.0935	0.0689	0.0649	0.0735	0.0956	0.0744	0.0684	0.0813
25	0.1222	0.0824	0.0686	0.0830	0.1341	0.0940	0.0848	0.0948
30	0.1228	0.0840	0.0894	0.0831	0.1221	0.0891	0.0952	0.0945
35	0.0988	0.1312	0.1091	0.1114	0.1085	0.1346	0.1099	0.1167

Tabla 6.83: *Comparación entre los resultados de la transformación lineal y PCA en Splice Rotados 8*

En la tabla comparativa se observa que en todos los casos es rentable hacer la reducción de la dimensionalidad con la transformación lineal, ya que no existe ningún

valor negativo. Cabe destacar que ejecutando el entrenamiento con 100 y 1.000 iteraciones los resultados son prácticamente idénticos. En el primer caso se obtiene un máximo valor de mejora de 13,12% y una mejora media de 9,03%, y en el segundo caso se alcanza un máximo de 13,46% y una mejora media de 9,16%.

En la siguiente ilustración se muestra una gráfica comparativa de los resultados obtenidos por la transformación lineal y PCA en el dominio *Splice rotados 8*. Nuevamente, se puede apreciar que los resultados de la transformación lineal son mejores en todos los casos de una forma amplia.

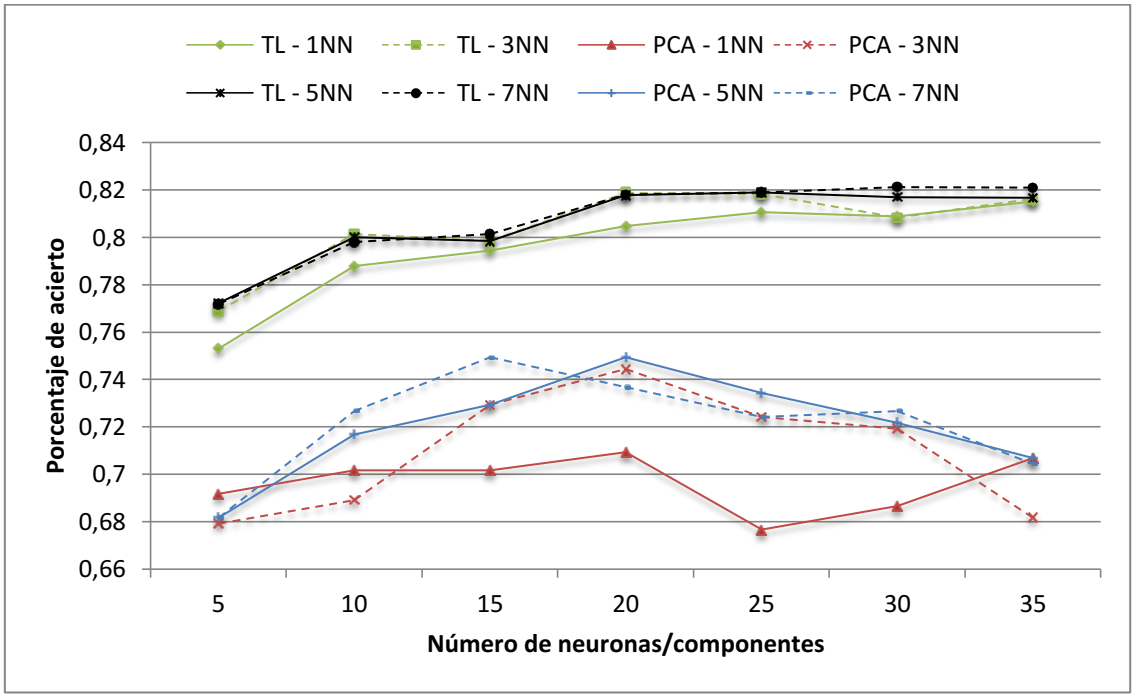

Ilustración 6.14: *Gráfica comparativa de la transformación lineal y PCA en Splice rotados 8*

Comparación entre los resultados de la transformación lineal normalizada y PCA

A continuación, se muestra una tabla comparativa entre la transformación lineal normalizada y PCA. Cabe destacar que los valores de la tabla han sido obtenidos calculando la resta de los resultados de la transformación lineal normalizada y PCA. Por este motivo, valores positivos en la tabla indicarán que se produce una mejora, mientras que valores negativos indican un empeoramiento.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
5	0.0612	0.0962	0.0988	0.0982	0.0631	0.0898	0.0932	0.0923
10	0.0802	0.1093	0.0857	0.0732	0.0706	0.0887	0.0682	0.0582
15	0.0839	0.0781	0.0836	0.0664	0.0746	0.0652	0.0677	0.0481
20	0.0802	0.0501	0.0496	0.0632	0.0723	0.0551	0.0583	0.0731
25	0.1037	0.0708	0.0639	0.0724	0.1140	0.0748	0.0773	0.0840
30	0.0898	0.0777	0.0748	0.0802	0.1042	0.0827	0.0813	0.0752
35	0.0749	0.1092	0.0927	0.0963	0.0830	0.1217	0.1027	0.1038

Tabla 6.84: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Splice Rotados 8*

Nuevamente, todos los resultados son positivos, por lo que la mejora que se produce utilizando la red de neuronas es clara. En primer lugar, realizando el entrenamiento con 100 iteraciones se obtiene una mejora media de 8,09% y se alcanza un máximo de 10,93%. Por otro lado, con 1.000 iteraciones el máximo logrado es 12,17% y la mejora media es de 8,01%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.85* se muestra la comparación realizada de los resultados obtenidos por la reducción de la dimensionalidad producida por la red de neuronas, aplicando la función sigmoidal, y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
5	0.0687	0.0847	0.0852	0.0867	0.0623	0.0798	0.0815	0.0810
10	0.0812	0.0933	0.0692	0.0562	0.0686	0.0822	0.0561	0.0466
15	0.0964	0.0760	0.0786	0.0601	0.0942	0.0632	0.0622	0.0421
20	0.0990	0.0661	0.0576	0.0677	0.0952	0.0615	0.0580	0.0695
25	0.1194	0.0730	0.0630	0.0714	0.1282	0.0844	0.0794	0.0861
30	0.1082	0.0815	0.0764	0.0748	0.1311	0.0920	0.0935	0.0892
35	0.1011	0.1345	0.1100	0.1136	0.1127	0.1421	0.1124	0.1135

Tabla 6.85: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Splice Rotados8*

En la tabla se observa que se mejoran todos los resultados de PCA y, por tanto, obtener la reducción de la dimensionalidad con la red de neuronas aplicando la función sigmoidal aporta grandes beneficios. Esta mejora es demostrada calculando la media, con 100 iteraciones se consigue un 8,41% y con 1.000 iteraciones un 8,46%. Merece la pena destacar también que los valores de mejora máximos logrados son 13,45% y 14,21%.

6.5.2.3. Análisis de los resultados obtenidos en los experimentos añadiendo 20 atributos adicionales y rotando los datos

En este apartado se van a analizar los resultados obtenidos en el dominio *Splice* habiendo añadido 20 atributos con valores calculados de forma aleatoria en el intervalo [0,1]. Por tanto, en este caso el conjunto de datos original tiene 80 atributos.

El porcentaje máximo de clasificación obtenido sobre los datos originales es de 71,93%.

En la *Tabla 6.76* se observa que con la transformación lineal se alcanza un resultado máximo de 82,96% con 25 neuronas en la capa oculta. Si comparamos este resultado con el porcentaje obtenido sobre el conjunto de datos original (71,93%), se puede apreciar como se produce una mejora de un 11,03%.

Con la transformación lineal normalizada también se produce mejora con respecto a la clasificación realizada sobre los datos originales (71,93%), puesto que se obtiene un valor máximo de 82,10% con 20 neuronas en la capa oculta. Es importante destacar que la mejora en la clasificación se consigue con datos con dimensionalidad reducida

utilizando únicamente 20 atributos frente a los 80 atributos que tiene el conjunto de datos original.

Por último, con la reducción de la dimensionalidad llevada a cabo aplicando la función sigmoidal se obtiene una tasa de acierto máxima de un 82,92% con 35 neuronas en la capa oculta. Como se puede observar, este valor mejora el porcentaje de acierto obtenido por KNN sobre los datos originales (71,93%). Por otro lado, cabe destacar que en este dominio la función sigmoidal no ofrece unos resultados tan buenos como la transformación lineal y la transformación lineal normalizada, ya que genera un conjunto de datos con un número de dimensiones sensiblemente mayor.

Adicionalmente, se observa en la *Tabla 6.79* que el mejor resultado que proporciona PCA es de 74,44% con 5 componentes. Nótese que se empeoran los resultados obtenidos realizando la reducción de la dimensionalidad con la transformación lineal, la transformación lineal normalizada y aplicando la función sigmoidal.

Comparación entre los resultados de la transformación lineal y PCA

En la *Tabla 6.86* se muestran el resultado de la comparativa obtenida con la transformación lineal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
5	0.0817	0.0561	0.0336	0.0215	0.1115	0.0777	0.0552	0.0409
10	0.0923	0.0927	0.0956	0.0885	0.0862	0.0847	0.0837	0.0802
15	0.0845	0.0916	0.1081	0.1269	0.0687	0.0742	0.0867	0.1058
20	0.1232	0.0952	0.1156	0.1024	0.1235	0.1046	0.1195	0.1078
25	0.1567	0.1286	0.1068	0.0824	0.1511	0.1163	0.0977	0.0751
30	0.1166	0.1115	0.1077	0.0919	0.1232	0.1161	0.1111	0.0881
35	0.1453	0.1437	0.1032	0.1148	0.1466	0.1424	0.1065	0.1169

Tabla 6.86: *Comparación entre los resultados de la transformación lineal y PCA en Splice Rotados 50*

Nuevamente, todos los resultados obtenidos son positivos por lo que en ningún caso se empeora el resultado de PCA. Cabe destacar que los resultados con 100 y 1.000 iteraciones son prácticamente idénticos, en el primer caso se logra un valor máximo de mejora de 15,67% y una mejora media de 10,07%, y en el segundo caso se consigue un máximo de 15,11% y un porcentaje medio de mejora de 10%.

Por último, se muestran en la *ilustración 6.15* los resultados de la transformación lineal y PCA en el dominio *Splice rotados 20*. En la gráfica es posible observar que los resultados de la transformación lineal son superiores con un margen grande.

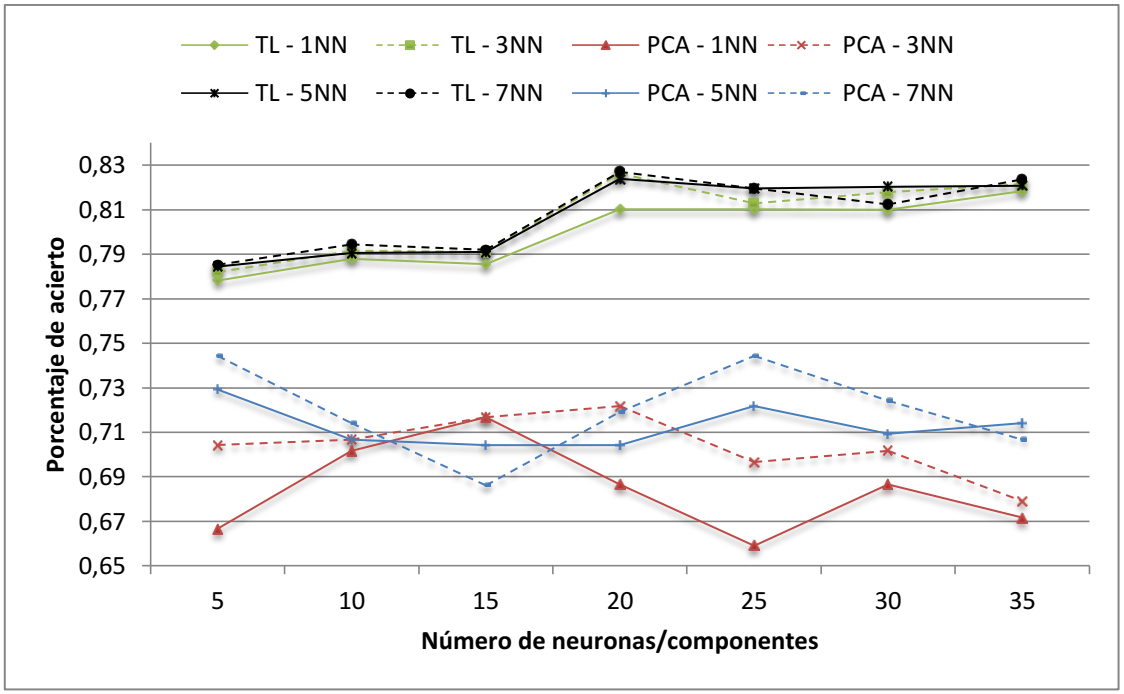

Ilustración 6.15: *Gráfica comparativa de la transformación lineal y PCA en Splice rotados 20*

Mediante el análisis de las distintas gráficas mostradas a lo largo de todos los apartados comparativos, se pone de manifiesto la clara superioridad que presenta la reducción de la dimensionalidad producida con la red de neuronas frente a PCA. Esto es debido a que en los dominios *Doughnut*, *Spambase* y *Splice* los resultados son mejores y solo en casos muy puntuales en el dominio *German* se empeoran los resultados.

Comparación entre los resultados de la transformación lineal normalizada y PCA

A continuación, se muestra la comparación hecha entre la transformación lineal normalizada y PCA en el conjunto de datos *Splice Rotados 20*.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
5	0.0837	0.0566	0.0451	0.0305	0.0889	0.0597	0.0447	0.0330
10	0.0609	0.0881	0.0944	0.0802	0.0807	0.0862	0.0862	0.0832
15	0.0827	0.0913	0.1059	0.1265	0.0516	0.0677	0.0897	0.1073
20	0.1117	0.0913	0.1059	0.0945	0.1189	0.0881	0.1167	0.1013
25	0.1279	0.1066	0.0840	0.0598	0.1263	0.0984	0.0727	0.0573
30	0.1024	0.1027	0.1015	0.0848	0.0998	0.0994	0.1011	0.0848
35	0.1307	0.1337	0.1044	0.1140	0.1245	0.1236	0.0898	0.1052

Tabla 6.87: *Comparación entre los resultados de la transformación lineal normalizada y PCA en Splice Rotados 50*

En la tabla comparativa se observa que se mejoran todos los resultados. Entrenando la red con 100 iteraciones la mejora media que se consigue es de 9,29% y el máximo de mejora que se registra es de 13,37%. Sin embargo, con 1.000 iteraciones se obtienen unos resultados ligeramente inferiores, ya que la mejora media alcanzada es de 8,88% y el máximo es 12,63%.

Comparación entre los resultados aplicando la función sigmoidal y PCA

En la *Tabla 6.88* se muestra la comparación entre la tasa de acierto media obtenida en la reducción de la dimensionalidad llevada a cabo por la red de neuronas, aplicando la función sigmoidal, y PCA. Los valores de la tabla han sido obtenidos calculando la resta de los resultados aplicando la función sigmoidal y PCA.

	100 iteraciones				1.000 iteraciones			
Vecinos / Elementos	1	3	5	7	1	3	5	7
5	0.0802	0.0506	0.0276	0.0110	0.1006	0.0614	0.0364	0.0229
10	0.0685	0.0693	0.0701	0.0643	0.0676	0.0656	0.0686	0.0611
15	0.0770	0.0845	0.0977	0.1122	0.0596	0.0576	0.0681	0.0897
20	0.1128	0.0831	0.1045	0.0891	0.1114	0.0781	0.0970	0.0813
25	0.1539	0.1197	0.0946	0.0745	0.1473	0.1109	0.0855	0.0626
30	0.1195	0.1061	0.1015	0,0848	0.1220	0.1061	0.0986	0.0856
35	0.1449	0.1408	0.1027	0.1090	0.1575	0.1449	0.1103	0.1132

Tabla 6.88: *Comparación entre los resultados aplicando la función sigmoidal y PCA en Splice Rotados50*

Por último, se observa que en este dominio con la función sigmoidal se mejoran todos los resultados de PCA. Con la red entrenada con 100 iteraciones se logra un aumento medio en la tasa de acierto de 9,12%, mientras que el mayor porcentaje registrado es 15,39%. Por otro lado, con 1.000 iteraciones la mejora media es de 8,83% y el máximo alcanzado en este caso es 15,75%.

6.6. Conclusiones del estudio realizado

En la *Tabla 6.89* se muestra un resumen con los mejores porcentajes de acierto obtenidos en los siguientes casos:

- **KNN:** Muestra los mejores resultados obtenidos por KNN sobre el conjunto de datos original, sin realizar reducción de la dimensionalidad.
- **TL-KNN:** Muestra los mejores resultados obtenidos sobre la reducción de la dimensionalidad producida por la transformación lineal. Además, se muestra entre paréntesis el número de atributos que contiene el conjunto de datos con el que ha sido obtenido dicho resultado.
- **TLN-KNN:** Muestra los mejores resultados obtenidos sobre la reducción de la dimensionalidad producida por la transformación lineal normalizada. Además, se muestra entre paréntesis el número de atributos que contiene el conjunto de datos con el que ha sido obtenido dicho resultado.
- **S-KNN:** Muestra los mejores resultados obtenidos sobre la reducción de la dimensionalidad producida aplicando la función sigmoidal. Además, se muestra entre paréntesis el número de atributos que contiene el conjunto de datos con el que ha sido obtenido dicho resultado.
- **PCA-KNN:** Muestra los mejores resultados obtenidos sobre la reducción de la dimensionalidad producida por PCA. Además, se muestra entre paréntesis el número de componentes que contiene el conjunto de datos con el que ha sido obtenido dicho resultado.

En la columna con el nombre #Atributos se muestra el número de atributos que tiene el conjunto de datos sin haber realizado la reducción de la dimensionalidad, es decir, el número de atributos que contiene el conjunto de datos original.

Dominio	Versión	#Atributos	KNN	TL-KNN	TLN-KNN	S-KNN	PCA-KNN
Spambase	Spambase	57	89,57%	92,92%(12)	88,26%(28)	93,20%(8)	91,85%(16)
	SpambaseRotados8	65	68,45%	92,21%/(2)	84,08%(38)	92,13%(2)	69,67%(26)
	SpambaseRotados50	107	58,91%	91,02%/(2)	86,13%(2)	92%(2)	55,38%(10)
Doughnut	Doughnut8	10	71,44%	99,31%(5)	99,07%(5)	99,50%(7)	63,13%(6)
	DoughnutRotados8	10	71,44%	99,39%(6)	99,07%(3)	98,35%(7)	63,13%(6)
German	German	24	71,50%	73,21%(12)	72,58%(12)	71,38%(2)	73,50%(14)
	GermanRotados8	32	71%	73,20%(4)	72,46%(10)	71,71%(2)	72%(10)
	GermanRotados24	48	69,25%	71,25%(18)	72,29%(24)	70,68%(20)	71,50%(6)
Splice	Splice	60	72,75%	83,41%(8)	82,88%(20)	82,28%(24)	79,25%(20)
	SpliceRotados8	68	68,92%	82,79%(15)	81,50%(10)	82,38%(35)	74,94%(10)
	SpliceRotados20	80	71,93%	82,86%(25)	82,10%(20)	82,92%(35)	74,44%(5)

Tabla 6.89: *Resumen con los mejores resultados obtenidos*

En la *ilustración 6.16* pueden verse los resultados de la *Tabla 6.89* para las tres versiones del dominio *Spambase*.

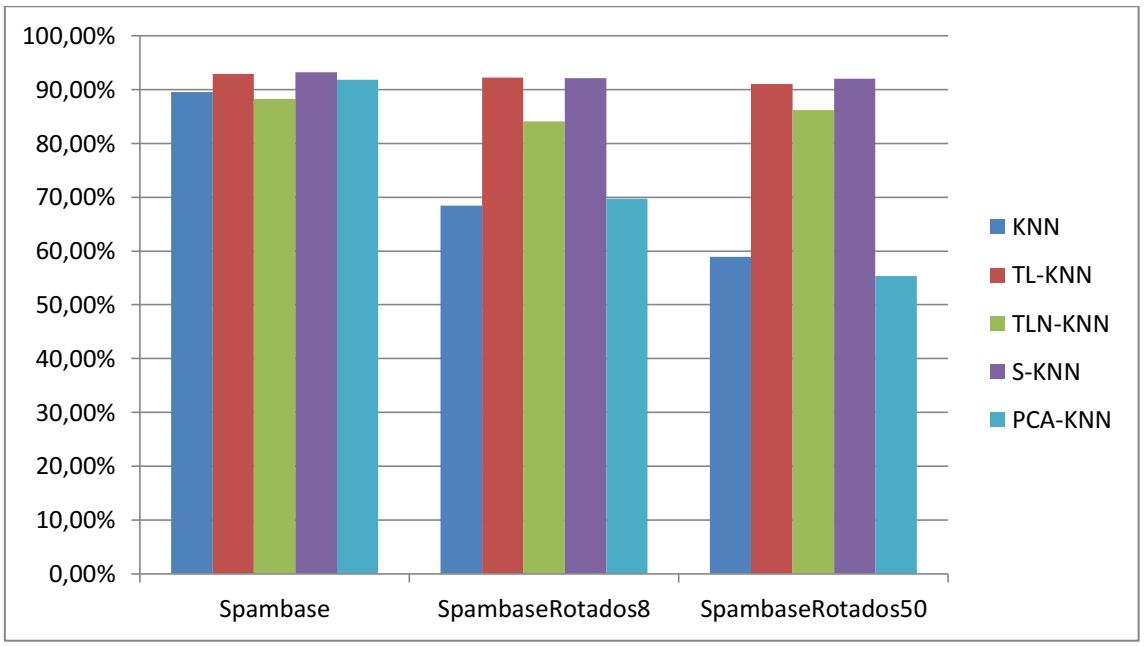

Ilustración 6.16: *Gráfica comparativa de los mejores resultados logrados por las distintas técnicas en el dominio Spambase*

Analizando los resultados del dominio *Spambase* se observa que sobre la versión de los datos sin modificar (en la tabla versión *Spambase*) se mejoran ligeramente los resultados obtenidos con TL-KNN, TLN-KNN y S-KNN, pero a medida que se aumenta el número de atributos aleatorios la mejora que se produce aumenta considerablemente. Sobre la versión de los datos *Spambase Rotados 8* y *Spambase Rotados 50* vemos como la reducción de la dimensionalidad que se produce es muy eficiente, llegando a superar el 90% en la clasificación (con la transformación lineal y aplicando la función sigmoidal) con un conjunto de datos con únicamente 2 atributos frente a los 65 atributos que tiene *Spambase Rotados 8* y los 107 atributos que tiene

Spambase Rotados 50. Teniendo en cuenta estos datos, se puede concluir que en este dominio es altamente rentable realizar la reducción de la dimensionalidad con la transformación lineal o aplicando la función sigmoidal.

En la *ilustración 6.17* se pueden observar los resultados de la *Tabla 6.89* para las dos versiones del dominio *Doughnut*.

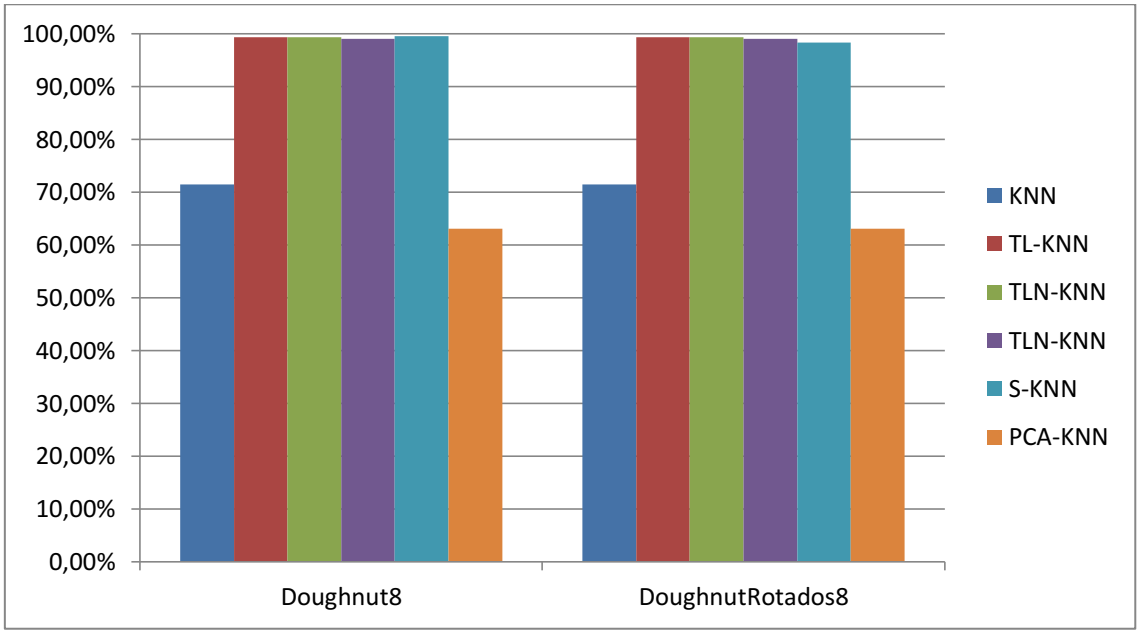

Ilustración 6.17: *Gráfica comparativa de los mejores resultados logrados por las distintas técnicas en el dominio Doughnut*

Las pruebas sobre el dominio conocido con el nombre *Doughnut* también arrojan muy buenos resultados con la reducción de la dimensionalidad realizada por la red de neuronas en cualquiera de sus tres versiones (TL, TLN o S). Podemos observar que tanto en la versión *Doughnut 8* como en *Doughnut Rotados 8* los resultados superan el 99% de acierto (excepto en un único caso que se obtiene 98,35%). Además, es importante destacar que utilizando menos atributos se obtiene un resultado que es mejor que obteniendo la clasificación sobre los datos originales. También se mejora ampliamente el resultado obtenido por PCA. Nuevamente, se puede concluir que en este dominio es muy rentable utilizar la reducción de la dimensionalidad con el modelo propuesto.

En la siguiente ilustración es posible observar los mejores resultados obtenidos por las distintas técnicas en el dominio *German*.

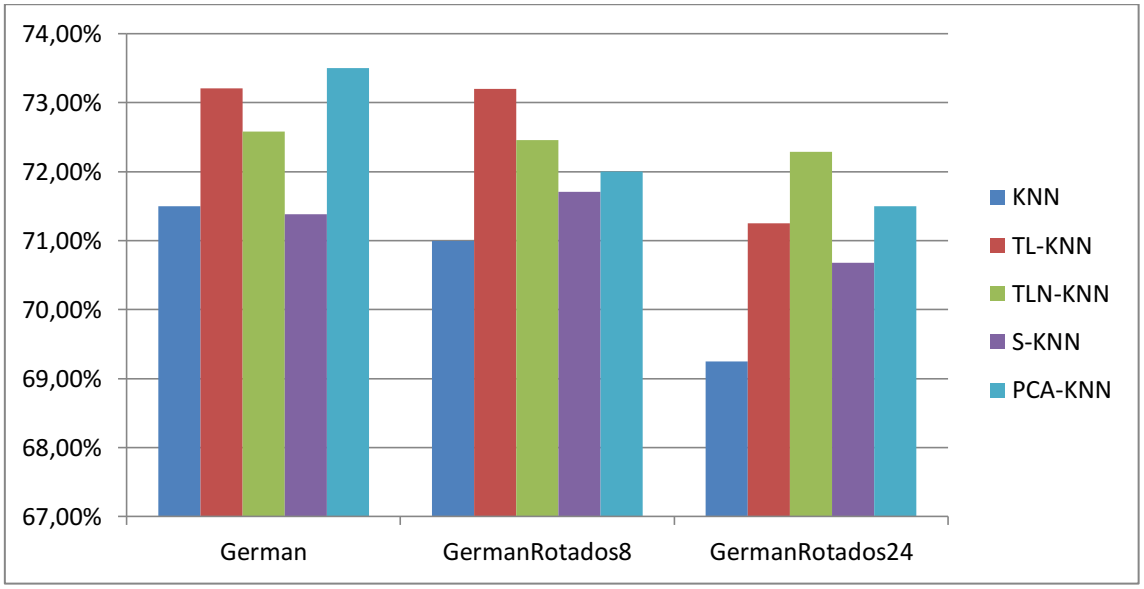

Ilustración 6.18: *Gráfica comparativa de los mejores resultados logrados por las distintas técnicas en el dominio German*

Sin embargo, en *German, German Rotados 8 y German Rotados 24* no se produce tanto beneficio mediante la reducción de la dimensionalidad llevada a cabo por la red de neuronas en cualquiera de sus tres versiones. En la *Tabla 6.89* se observa como se mejora ligeramente la tasa de acierto obtenida sobre los datos originales (solo en *German* con S-KNN se empeora el resultado), pero esta mejora no es significativa. Además, parece que la transformación lineal en este caso es un poco más estable y ofrece un resultado más elevado que la transformación lineal normalizada y la sigmoidal. Por tanto, en este dominio, aunque la mejora no es tan amplia, también es rentable realizar la reducción de la dimensionalidad utilizando la red de neuronas.

En la *ilustración 6.19* se pueden apreciar los resultados mostrados en la *Tabla 6.89* en el dominio *Splice*.

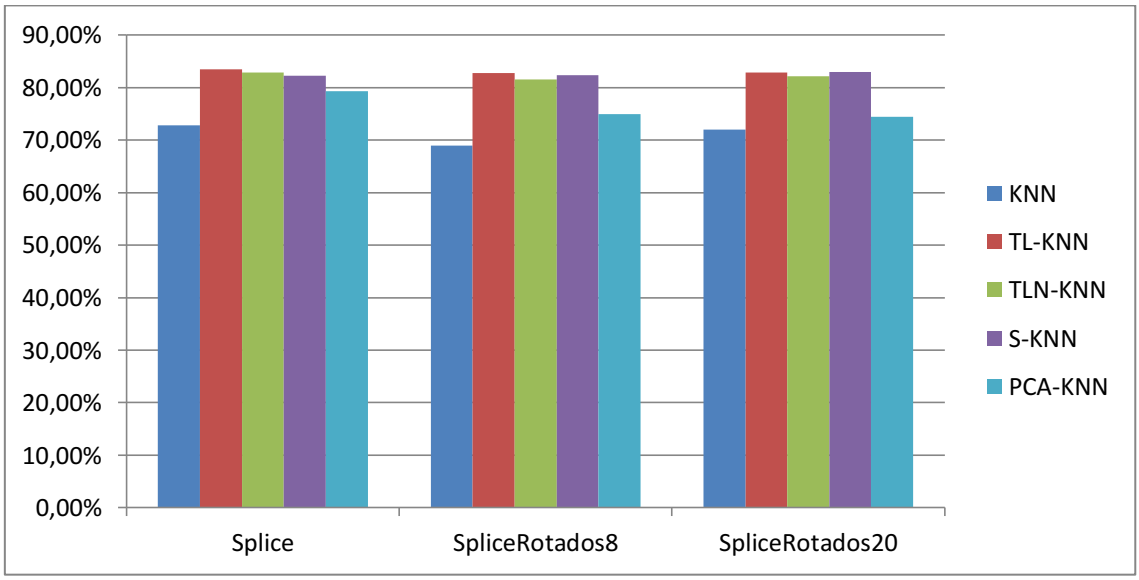

Ilustración 6.19: *Gráfica comparativa de los mejores resultados logrados por las distintas técnicas en el dominio Splice*

Por último, en *Splice*, *Splice Rotados 8* y *Splice Rotados 20* se observa que nuevamente es aconsejable realizar la reducción de la dimensionalidad utilizando la red de neuronas, ya que en este dominio también se mejoran en todos los casos los resultados obtenidos en la clasificación de los datos originales. Adicionalmente, la transformación lineal parece que también en este caso es más estable y ofrece unos mejores resultados que la transformación lineal normalizada y la sigmoidal.

Resumen de la comparación de la reducción de la dimensionalidad realizada por la red de neuronas y PCA

En las siguientes tablas se muestra el valor máximo de mejora registrado en la comparación, así como la mejora media que se obtiene. En la tabla valores negativos indican una pérdida de precisión con respecto a PCA, por el contrario si se obtienen valores positivos indica que se mejoran los resultados.

Dominio	Versión	TL vs. PCA		TLN vs.PCA		S vs. PCA	
		Máximo	Media	Máximo	Media	Máximo	Media
Spambase	Spambase	4,97%	1,69%	-2,94%	-9,17%	5,52%	1,86%
	SpambaseRotados8	38,35%	24,98%	28,12%	16,83%	40,53%	31,34%
	SpambaseRotados50	40,38%	31,25%	35,64%	31,25%	40,16%	32,76%
Doughnut	Doughnut8	44,61%	32,96%	44,61%	32,96%	37,15%	25,29%
	DoughnutRotados8	44,92%	33,30%	44,82%	33,18%	36,76%	25,81%
German	German	6,43%	1,84%	5,50%	1,17%	4,70%	5,13%
	GermanRotados8	8,29%	2,56%	8,63%	2,28%	8,46%	1,35%
	GermanRotados24	6,54%	3,07%	5,43%	3%	7,04%	2,66%
Splice	Splice	9,75%	5,71%	8,33%	4,43%	9,52%	4,56%
	SpliceRotados8	13,12%	9,03%	10,93%	8,09%	13,45%	8,41%
	SpliceRotados20	15,67%	10,07%	13,37%	9,29%	15,39%	9,12%

Tabla 6.90: *Resumen de la comparación de PCA con la red entrenada con 100 iteraciones*

Dominio	Versión	TL vs. PCA		TLN vs.PCA		S vs. PCA	
		Máximo	Media	Máximo	Media	Máximo	Media
Spambase	Spambase	5%	1,75%	-5,28%	-11,1%	5,61%	1,76%
	SpambaseRotados8	38,15%	25,03%	27,22%	15,08%	38,28%	26,45%
	SpambaseRotados50	40,53%	31,34%	31,28%	21,03%	40,61%	32,98%
Doughnut	Doughnut8	45,66%	34,45%	45,25%	34,52%	41,75%	28,21%
	DoughnutRotados8	43,53%	33,48%	45,35%	33,6%	41,22%	28,24%
German	German	5,92%	1,99%	6,50%	1,32%	5,63%	0,57%
	GermanRotados8	8,50%	2,96%	8,60%	2,15%	8,40%	2,14%
	GermanRotados24	6,40%	2,79%	6,50%	3,29%	7,80%	3%
Splice	Splice	10,47%	5,37%	8,56%	4,20%	9,63%	4,28%
	SpliceRotados8	13,46%	9,16%	12,17%	8,01%	14,21%	8,46%
	SpliceRotados20	15,11%	10%	12,63%	8,88%	15,75%	8,83%

Tabla 6.91: *Resumen de la comparación de PCA con la red entrenada con 1.000 iteraciones*

En las anteriores tablas se observa que solo en el caso de *Spambase* con la TLN se obtiene una pérdida en la tasa de acierto, en el resto de casos se consiguen resultados positivos. También es importante observar que la mejora en muchos casos es muy elevada, así por ejemplo en el dominio *Doughnut* se logra en algunos casos un máximo superior a un 40% de mejora y la media se supera en más de un 30%. Teniendo en

cuenta los datos de la tabla, se puede concluir que la mejora que se produce con respecto a PCA es muy elevada.

Como conclusión comentar que cualquiera de las tres versiones estudiadas (transformación lineal, transformación lineal normalizada y sigmoidal) parece ser viable como técnica de reducción de dimensionalidad. En los dominios en los que se ha realizado el estudio, la transformación lineal ha ofrecido unos resultados ligeramente mejores de forma genérica. A pesar de esto, no se puede recomendar este método por encima de la transformación lineal normalizada o la sigmoidal y se recomienda realizar un estudio previo con las tres versiones, eligiendo la que mejor resultados proporcione.

Capítulo 7

Conclusiones

7.1. Objetivos cumplidos

En el capítulo 2 se describen las técnicas existentes de reducción de la dimensionalidad, así como los problemas que presentan los conjuntos de datos con un número elevado de dimensiones. Además, en este capítulo se presentan trabajos similares de estudios realizados en el ámbito de la reducción de la dimensionalidad y algunas aplicaciones.

Como se detalla en el capítulo 6, ha sido implementado un script con el lenguaje de programación R para llevar a cabo los experimentos de reducción de la dimensionalidad con la red neuronal, en este script también se incluye una función que permite extraer las activaciones de las neuronas de la capa oculta. Adicionalmente, se detalla como ha sido creado otro script para aplicar KNN sobre los datos originales y obtener los resultados de clasificación sobre la reducción de la dimensionalidad realizada utilizando PCA. Cabe destacar la existencia de funciones en ambos script para realizar un preprocesado de los datos, concretamente se aplica la normalización, aleatorización y separación en entrenamiento y test. La numerización ha sido aplicada previamente.

Sin embargo, el principal objetivo del proyecto era estudiar la viabilidad de varias técnicas de reducción de la dimensionalidad basadas en el Perceptrón Multicapa, esto es llevado a cabo de forma empírica en el capítulo 7. En este capítulo, además de presentar los resultados de los experimentos, se muestra una comparación de la eficiencia en la clasificación de los datos originales con la eficiencia de la clasificación de las proyecciones obtenidas por la red neuronal. En dicho capítulo también se lleva a cabo la comparación con PCA.

Al ver los buenos resultados que ofrecía el método de reducción de dimensionalidad estudiado, además de los objetivos iniciales, se propuso la creación de una librería para que este método pudiese ser usado por la comunidad de forma libre. Por tanto, se decidió finalmente crear un nuevo modelo para la librería llamada *Caret*. Dicho modelo lleva a cabo una reducción de la dimensionalidad de los datos antes de realizar la tarea de clasificación con KNN. El proceso de creación de este modelo, así como ejemplos de su uso son mostrados en el final del capítulo 5.

7.2. Líneas de trabajo futuras

Debido a que se trata de un trabajo experimental de investigación, ha habido algunas opciones que no han podido ser exploradas por tener un excesivo coste en tiempo. Por tanto, en este apartado se describen algunos de los posibles trabajos futuros que se pueden llevar a cabo.

Una línea que queda abierta es realizar el mismo estudio que el llevado a cabo en este documento utilizando un clasificador distinto a KNN para hacer la evaluación, ya que los resultados obtenidos por este algoritmo son satisfactorios y, sin embargo, con otros algoritmos de clasificación se podría obtener un resultado negativo.

Por otro lado, la principal línea de trabajo que queda abierta es realizar el estudio de la viabilidad de la reducción de la dimensionalidad empleando una red neuronal con más de una capa oculta. En este caso habría que estudiar los resultados que se obtienen en la clasificación de las activaciones de las neuronas de la última capa oculta. Para realizar la evaluación se podría utilizar también el algoritmo KNN u otro algoritmo cualquiera de clasificación.

Bibliografía

[Afken, 1985] ARFKEN, G., 1985. *Mathematical Methods for Physicists*. 3ª edición. Orlando: Academic Press. ISBN 9780120598762.

[Agrawal y Srikant, 1994] AGRAWAL, Rakesh, SRIKANT, Ramakrishnan, 1994. Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco: Morgan Kaufmann Publishers, páginas 487-499. ISBN 1558601538.

[Bellman y Corporation, 1957] BELLMAN, Richard Ernest, CORPORATION, Rand 1957. *Dynamic programming*. Princeton University Press. ISBN 9780691079516.

[Bellman, 1961] BELLMAN, Richard Ernest, 1961. *Adaptative control processes: a guided tour*. 5ª edición. Princeton University Press. ISBN 9780691625850.

[Borrajo, 2006] BORRAJO, Daniel, 2006. *Aprendizaje automático*. Madrid: Sanz y Torres. ISBN 8496094731.

[Chung, 2006] CHUNG, Hiu, 2006. *Clustering, Dimensionality reduction, and Side Information* [en línea]. Tesis doctoral. Michigan State University [consulta en: Junio de 2017]. Disponible en: http://biometrics.cse.msu.edu/Publications/Thesis/MartinLaw_Clustering_PhD06.pdf

[Dy et al., 2003] DY, J.G., BRODLEY, C.E., KAK, A.C., BRODERICK, L.S., AISEN, A., 2003. Unsupervised feature selection applied to content-based retrieval of lung images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volumen 25, páginas 373-378.

[Golub et al., 1999] GOLUB, T.R., SLONLM, D.K., TAMAYO, P., HUARD, C., GAASENBEEK, M., MESIROV, J.P., COLLER, H., LOH, M.L., DOWING, J.R., CALIGIURI, M.A., BLOOMFIELD, C.D., LANDER, E.S., 1999. Molecular Classification of Cancer: Class Discovery and Clas Prediction by Gene Expression Monitoring. En: *Science* [en línea]. New York: Science, volumen 286, páginas 531-537 [consulta: Mayo de 2017]. DOI 10.1126/science.286.5439.531. Disponible en: http://isites.harvard.edu/fs/docs/icb.topic1013123.files/Golub%20et%20al%201999%20Cancer%20Classification%20using%20microarrays.pdf

[Goodfellow et al., 2016] GOODFELLOW, Ian, et al., 2016. *Deep Leaning* [en línea]. MIT Press [consulta: Mayo de 2017]. Disponible en: http://www.deeplearningbook.org/

[Guyon et al., 2005] GUYON, I., GUNN, S., BEN-HUR, A.y DROR, G., 2005. Result analysis of the NIPS 2003 feature selection challenge. En Advances in Neural Information Processing Systems 17, páginas 545–552 [en línea]. Disponible en: https://papers.nips.cc/paper/2728-result-analysis-of-the-nips-2003-feature-selection-challenge.pdf [consulta: Mayo de 2017]

[Hernandéz, Ramirez y Ferri, 2004] HERNÁNDEZ, J., RAMÍREZ, M.J., FERRI, C, 2004. *Introducción a la Minería de Datos*. Madrid: Pearson Educación, S.A. ISBN 84-205-4091-9.

[Hernando, 2013] HERNANDO, Pedro José, 2013. *Clases de Álgebra Lineal para Imgeniería*. 3.5ª revisión. Pedro José Hernando Oter. ISBN 9788469189344.

[Hotelling, 1933] HOTELLING, Harold, 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, volumen 24, páginas 417-441 y 498-520.

[Isasi y Galván, 2003] ISASI, Pedro, GALVÁN, Inés M., 2003. *Redes de neuronas artificiales. Un enfoque práctico*. Madrid: Pearson Educación. ISBN 8420540250.

[LeCun et al., 1990] LECUN, Y., MATAN, O., BOSER, B., DENKER, J.S., HENDERSON, D., HOWARD, R.E., HUBBARD, W., JACKEL, L.D., BAIRD, H.S., 1990. Handwritten zip code recognition with multilayer networks. *10th International Conference on Pattern Recognition*, volumen 2, páginas 35–40.

[Lee et al., 2000] LEE, W., STOLFO, S.J., MOK, K.W., 2000. Adaptative intrusion detection: A data mining approach. *AI Review*, volume 14, páginas 533-567.

[Lodha y Kampalur, 2014] LODHA, S.P., KAMPALUR, S.M., 2014. Dimensionality Reduction Techniques for Hyperspectral Images. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)* [en línea], volumen 3, páginas 92-99 [consulta en: Mayo de 2017]. ISSN 2319-4847. Disponible en: http://www.ijaiem.org/Volume3Issue10/IJAIEM-2014-10-24-51.pdf

[Mateo, 2012] MATEO, Fernando, 2012. *Redes neuronales y preprocesado de variables para modelos y sensors en bioingeniería* [en línea]. Valencia: Universitat Politècnica de València [consulta en: Junio de 2017]. Disponible en: https://riunet.upv.es/bitstream/handle/10251/16702/tesisUPV3874.pdf?sequence=1

[Pearson, 1901] PEARSON, Karl, 1901. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine* [en línea], volumen 2, número 11, páginas 559-572. DOI 10.1080/14786440109462720 [consulta: Mayo de 2017]. Disponible en: http://stat.smmu.edu.cn/history/pearson1901.pdf

[Rignèr, 2008] RIGNÈR, Markus, 2008. What is principal component analysis?. *Nature biotechnology* [en línea], volumen 26, número 3, páginas 303-304. DOI 10.1038/nbt0308-303 [consulta en: Mayo de 2017]. Disponible en: http://www.nature.com/nbt/journal/v26/n3/full/nbt0308-303.html

[Segovia, 2015] SEGOVIA, Ignacio, 2015. *Estudio de técnicas de reducción de dimensionalidad* [en línea]. Trabajo fin de grado. Leganés: Universidad Carlos III de Madrid [consulta: Mayo de 2017]. Disponible en: http://e-archivo.uc3m.es/handle/10016/23043

[Shlens, 2003] SHLENS, Jonathon, 2003. A tutorial on Principal Component Analysis. En: *Cornell University Library* [en línea]. Disponible en: https://arxiv.org/abs/1404.1100 [consulta: Mayo de 2017]

[Shylaja, Balasubramanya y Natarajan, 2011] SHYLAJA, S, BALASUBRAMANYA, M., NATARAJAN, S., 2011. *Dimensionality Reduction Techniques for Face Recognition, Reviews, Refinements and New Ideas in Face Recognition*. Dr. Peter Corcoran (Ed.). InTech [consulta en: Mayo de 2017]. DOI 10.5772/18251. Disponible en: https://www.intechopen.com/books/reviews-refinements-and-new-ideas-in-face-recognition/dimensionality-reduction-techniques-for-face-recognition

[Swets y Weng, 1995] SWETS, D.L., WENG, J.J., 1995. Efficient content-based image retrieval using automatic feature selection. *IEEE Internetional Symposium or Computer Vision*, páginas 85-90.

[Wack et al., 2006] WACK, N., CANO, P., DE JONG, B., MARXER, R., 2006. A Comparative Study of Dimensionality Reduction Methods: The Case of Music Similarity. En: *Universitat Pompeu Fabra Barcelona* [en línea]. Disponible en: http://mtg.upf.edu/node/1955 [consulta: Mayo de 2017]

[Yang y Pederson, 1997] YANG, Y., PEDERSON, J.O., 1997. A comparative study on feature selection in text categorization. *Proceedings of the Fourteenth International Conference on Machine Learning*, páginas 412-420.

Anexo I

Abstract

1. Introduction

At present, multiple problems can be found where the dimensionality of the data is very high and this trend is on the rise. These types of problems pose a challenge in different areas of research, in this case all efforts will focus on reducing the number of data dimensions to improve the efficiency of machine learning algorithms. Performing this task in many domains is highly recommended because such algorithms function very badly as data dimensions increase and drastically reduce their efficiency in solving problems with these characteristics.

One of the most widely used and known techniques for dimensionality reduction is Principal Component Analysis (PCA). This technique performs the transformation of the data by generating new components, called main components, that are calculated taking into account the variance of the data set. However, it has the main problem that is an unsupervised technique, that is, it doesn't use the expected value for each data. For this reason, PCA can provide unsatisfactory results in some domains.

In order to solve the PCA problems and to make a better transformation of the data to a set with smaller dimensions in supervised problems, a scheme based on artificial neural networks is proposed, by using the model that is called Multi-layer Perceptron (MLP). This model is formed by artificial neurons that are organized in different types of layers.

In the Multi-layer Perceptron, the first layer, also called the input layer, contains neurons that are responsible for distributing the input data to the neurons of the next layer called the hidden layer. The neurons in the hidden layer calculate their activation level based on the information they receive and, in turn, propagate it to neurons in the next layer. The last layer of the network, known as the output layer, contains the neurons that give a response to the outside.

We can consider that the hidden layer performs a transformation of the input space of the original data with n dimensions to a different space with m dimensions. If m is less than n, a reduction in dimensionality will have occurred. This transformation of n to m dimensions may be linear or non-linear: the values that the hidden neurons receive from the input layer correspond to a linear transformation. The hidden neurons will calculate their activations by applying to these values a non-linear function. These activations correspond to the non-linear transformation of m dimensions.

Taking into account the previous reasoning, the architecture of the neural network proposed to perform the reduction of the dimensionality will be formed by an input

layer, a hidden layer and an output layer. In the input layer there will be as many neurons as dimensions have the original data set and the reduction of dimensionality will occur since, once the training of the network is finished, the activations of neurons of the hidden layer will be used as the attributes of the new dataset. Therefore, the number of neurons in the input layer must be strictly greater than the number of neurons in the hidden layer in order to carry out the reduction properly. It is important to highlight that, in the training phase, a single neuron will be used in the output layer that will provide the expected value for each pattern.

In addition, a comparison will be made between the two previous techniques to see if the results of PCA in supervised classification problems are improved. In order to compare and evaluate the quality of each solution a machine learning algorithm, known as K-nearest neighbors (KNN), will be used.

2. Goals

The main goal of this project is to propose a supervised dimensionality reduction method based on artificial neural network and carry out an exhaustive experimental study to evaluate this method with different alternatives and parameters.

In particular, the specific objectives that are proposed for the present work are the following:

- To perform a state-of-the-art study of existing methods of dimensionality reduction.
- Implementation of an artificial neural network by using the Multilayer Perceptron for classification problems. The network must be trained and it will be necessary to implement a method to obtain the activations of neurons of the hidden layer.
- Implementation of PCA.
- Empirical study of the reduction of the dimensionality performed by the hidden layer of the Multilayer Perceptron. In this study, it will be verified if, by classifying the data with reduced dimensions, the classification results of the original data are improved. In addition, a comparison will be made with the results provided by PCA.
- If the results obtained in the experimentation are good, a library will be created with the dimensionality reduction method studied. This library can be used freely by the community.

3. Methodology

In this section we will explain in detail the procedure chosen to perform the analysis of the method of dimensionality reduction based on Multilayer Perceptron, as well as the comparison with PCA.

Firstly, the original data of the domains used in the study have been subjected to a preprocessing of data before using the dimensionality reduction techniques. The techniques employed in this work have been the normalization and randomization of the data.

With the data prepared, it is possible to begin to apply the procedure for the reduction of dimensionality. This process can be divided into three blocks:

- Phase 1: Reduction of dimensionality by applying the Multi-layer Perceptron and evaluation by using KNN.
- Phase 2: Evaluation of the accuracy of KNN on the original data.
- Phase 3: Reduction of dimensionality by applying PCA and evaluation by using KNN.

It must be taken into account that for each domain different experiments will be done, by reducing the number of dimensions in different sizes to find the optimal value of dimensions in each case. In addition, as a comparative study will be done with PCA in each domain, experiments will be performed with the same number of components (in the case of PCA) and neurons in the hidden layer (in the case of the scheme that uses the Multi-layer Perceptron).

In addition, the parameter *k* of the KNN algorithm will be obtained experimentally, that is, by trial and error. For this reason, different tests will be performed by varying this parameter, the values of *k* used in each domain will be 1, 3, 5 y 7.

3.1. Phase 1: Reduction of dimensionality by applying the Multi-layer Perceptron and evaluation by using KNN

3.1.1. Network design

The first step to use a model based in Multi-layer Perceptron is to define its architecture. In this case, the design choices are as follows:

- The number of neurons in the input layer will be equal to the number of attributes that the data set has, that is, each attribute will correspond with an input neuron.
- There will be only one output neuron. In this case, the problems to solve will be of biclass classification, so the output of the Multi-layer Perceptron should be interpreted as follows: if the result of the network is in the interval [0, 0.5], it will be considered that the data belongs to the first class; whereas if the result is in the interval (0.5, 1], it will be considered that the data belongs to the second class.
- There will be a single hidden layer. The number of neurons in the hidden layer will correspond to the number of dimensions of the new data set transformed. In each domain experiments with different numbers of neurons in the hidden layer will be performed to find the optimal value.
- The activation function chosen for all neurons is the sigmoidal function.

3.2.1. Network training

The training of the neural networks based on Multi-layer Perceptron will be done by using the Backpropagation algorithm. In this case, the stopping criterion used to stop learning consists of previously establishing a fixed number of iterations, so that to avoid underfitting and overfitting of the model, two independent trainings will be performed by using different numbers of iterations in each training. Therefore, the same experiments will be performed with 100 and 1.000 training iterations.

3.3.1. Evaluation of the neural network

Once the network of neurons is trained, the next step will be to measure the accuracy rate on the training data in the output of the network.

3.4.1. Evaluation of the activations of hidden layer neurons

As the attributes of the new dataset correspond to the activations of neurons of the hidden layer, KNN must be applied to the activations of said neurons to carry out the evaluation of the dimensionality reduction done. In this case, it has been decided to perform three different evaluations and are the following:

1. Evaluation of the linear transformation performed before applying the sigmoidal function of the hidden layer.
2. Evaluation of the normalized linear transformation.
3. Evaluation of the transformation performed by the hidden layer, when the sigmoidal function has been applied.

Next, a detailed description of the procedure required to carry out the three evaluations will be done.

Evaluation of the linear transformation performed before applying the sigmoidal function of the hidden layer

After the training of the network, the linear transformation performed by the hidden layer of the network will be calculated. To obtain such activations, the following equation must be solved:

$$Y = XW^T \tag{8.1}$$

Where:

- Y is the matrix that contains the linear transformation performed by the neurons of the hidden layer, that is, it contains the data with a reduced number of dimensions.
- W is the matrix that contains the bias of each neuron of the hidden layer and the weights of the connections between the neurons of the input layer and the neurons of the hidden layer.
- X is the matrix that contains the original data and an additional column with the value 1, this column is added to be able to apply the bias of the neurons.

It is important to note that the activations that produce the training data and the activations that produce the test data are calculated separately. Hence, equation (8.1) must be applied twice, on the one hand, the matrix X corresponds to the training data matrix X_e which generates a new matrix Y_e and, on the other hand, the matrix X corresponds to the test data matrix X_t generating another matrix Y_t.

Once the matrix Y is calculated for the training data and the test data, the next step is to perform the evaluation by utilizing KNN that, by using the projections of the training data (matrix Y_e), will do a classification for each projection of the test set (each row of the matrix Y_t). In this process, n different percentages of success are obtained. The accuracy of the transformation will be evaluated by the average of the n percentages obtained.

Evaluation of the normalized linear transformation

The next step is to obtain two new matrices which we will denote as Y_e' and Y_t', it will be the result of applying the normalization in the interval [0,1] of the matrices Y_e and Y_t, respectively.

Again, the result of this transformation should be evaluated by KNN. The data of the matrix Y_e' will be used to perform the classification of all test data contained in the matrix Y_t', thus obtaining *n* percentages of success. After that, the average rate of success will be calculated to evaluate the model.

Evaluation of the transformation performed by the hidden layer, when the sigmoidal function has been applied

The next step will be to apply the sigmoidal activation function, both for the matrix Y_e and for the matrix Y_t. The sigmoidal function must be applied to each element of Y_e and Y_t, by generating two new matrices S_e and S_t.

The sigmoidal function must be applied to each element of Y_e and Y_t. To do this, the following equation must be used:

$$s_{ij} = \frac{1}{1+e^{-y_{ij}}} \tag{8.2}$$

Where:

- s_{ij} is the element of row *i* and column *j* of matrix *S*.
- y_{ij} is the element of row *i* and column *j* of matrix *S*.

Analogous to the previous cases, the average success rate obtained by KNN will be used to evaluate the transformation. In this case, matrix S_e will be used to classify all the data for S_t.

This process is repeated 50 times, having previously fixed the parameter *k* for KNN, the number of neurons in the hidden layer and the number of training iterations. First, 10 repetitions are made by storing all the results obtained in the following cases:

- Success rate obtained in the exit of the network for the training set.
- Success rate obtained by KNN by performing the classification of the calculated projections with the linear transformation.
- Success rate obtained by KNN by classifying the projections calculated with the normalized linear transformation.
- Success rate obtained by KNN by performing the classification of the calculated projections when the sigmoidal function has been applied.

Once the first 10 repetitions are executed, the best one is chosen. To do this, the repetition with the highest success rate on the training set is chosen. For the best repetition, the success rates of the three dimensionality reduction versions are saved (linear transformation, normalized and sigmoidal linear transformation). This process is repeated for 5 times, thus obtaining 5 executions. Finally, the final hit rate for the three approximations is calculated as the average of the 5 executions obtained.

3.2. Phase 2: Evaluation of the accuracy of KNN on the original data

In each domain the efficiency of the KNN classifier will be measured, by applying it to the original data. To do this, the original training set X_e will be used as training samples and a classification will be made for each data of the original test set X_t, with the average success rate being calculated for the evaluation.

3.3. Phase 3: Reduction of dimensionality by applying PCA and evaluation by using KNN

In this phase, PCA will be used to perform the reduction of the dimensionality. As in the case of the Multi-layer Perceptron, different experiments will be performed in each domain to find the optimal value of components in each case.

The first step consists in calculating the main components by using only the training data matrix X_e. Once all the main components are calculated, those that most variance explain are chosen and the matrix *U* is formed to perform the linear transformation. To do this, it is necessary to solve the following equation:

$$Y = UX^T \tag{8.3}$$

Where:

- *Y* is the matrix with the data with a reduced number of dimensions.
- *U* is the matrix that contains the best *k* main components.
- *X* is the matrix that contains the original data.

Finally, the quality of the transformation performed by PCA is measured by using KNN, that is, with the projections of the training data (matrix Y_e), a classification is made for each data of the test set with reduced dimensions (each row of the matrix Y_t) and the average hit rate is calculated.

4. Results obtained

To perform the empirical study, the methodology described in the previous section has been used in four different domains. The domains used are the following:

- *Doughnut*: This data set has been artificially generated, that is, the data doesn't come from the real world. Initially this data set has been created with two attributes and the class of each pattern. The class can take the values: TRUE and FALSE. This domain is formed by 10.000 instances.
- *Spambase*: This domain, donated by George Forman, contains a database of emails that have been labeled as desired mail or spam. A total of 4.601 emails have been registered. The data set is formed by 57 attributes that contain essentially whether a word or character is frequently encountered in an e-mail.
- *German*: This data set has been donated by Hans Hofmann, professor at the University of Hamburg, and contains a database of users who are classified as good or bad clients. The data set is formed by 1.000 instances and 24 attributes.

- *Splice*: This domain contains a data set that is formed by 60 attributes with information relating to a DNA sequence. In addition, each instance contains its corresponding class that can take two possible values: if the instance is recognized as an exon/intron boundary or, conversely, if it is recognized as an intron/exon. The data set contains a total of 1.000 instances.

In real domains, it is common to find attributes that contribute nothing in solving the problem and may even lead to poorer results. To simulate this situation and to verify if the neural network is able to extract the relevant information of each problem, in each domain experiments have been carried out by adding additional attributes whose values have been calculated totally randomly. In addition, in some cases a rotation of the data will be performed to distribute the noise of the random attributes among all the new attributes. If the rotation is not done, there will be the original attributes with useful information and attributes added with irrelevant information or noise. With the rotation, everything is mixed, and there will be noise in all the attributes.

As a large number of results have been obtained throughout this study, this section only shows a summary of these results. Subsequently, an analysis of this summary of data is carried out.

The below table shows a summary with the best percentage of success obtained in the following cases:

- KNN: It shows the best results obtained by KNN on the original data set, without reducing the dimensionality.
- TL-KNN: It shows the best results obtained on the reduction of the dimensionality carried out by the linear transformation. In addition, the number of attributes with which the result was obtained is shown in parentheses.
- TLN-KNN: It shows the best results obtained on the reduction of the dimensionality carried out by the normalized linear transformation. In addition, the number of attributes with which the result was obtained is shown in parentheses.
- S-KNN: It shows the best results obtained on the reduction of the dimensionality carried out by applying the sigmoidal function. In addition, the number of attributes with which the result was obtained is shown in parentheses.
- PCA-KNN: It shows the best results obtained on the reduction of the dimensionality performed by PCA. In addition, the number of components contained in the dataset with which the result was obtained is shown in parentheses.

The column named #Attributes shows the number of attributes that the data set has without performing the dimensionality reduction, that is, the number of attributes contained in the original data set.

Domain	Version	#Attributes	KNN	TL-KNN	TLN-KNN	S-KNN	PCA-KNN
Spambase	Spambase	57	89,57%	92,92%(12)	88,26%(28)	93,20%(8)	91,85%(16)
	SpambaseRotated8	65	68,45%	92,21%/(2)	84,08%(38)	92,13%(2)	69,67%(26)
	SpambaseRotated50	107	58,91%	91,02%/(2)	86,13%(2)	92%(2)	55,38%(10)
Doughnut	Doughnut8	10	71,44%	99,31%(5)	99,07%(5)	99,50%(7)	63,13%(6)
	DoughnutRotated8	10	71,44%	99,39%(6)	99,07%(3)	98,35%(7)	63,13%(6)
German	German	24	71,50%	73,21%(12)	72,58%(12)	71,38%(2)	73,50%(14)
	GermanRotated8	32	71%	73,20%(4)	72,46%(10)	71,71%(2)	72%(10)
	GermanRotated24	48	69,25%	71,25%(18)	72,29%(24)	70,68%(20)	71,50%(6)
Splice	Splice	60	72,75%	83,41%(8)	82,88%(20)	82,28%(24)	79,25%(20)
	SpliceRotated8	68	68,92%	82,79%(15)	81,50%(10)	82,38%(35)	74,94%(10)
	SpliceRotated20	80	71,93%	82,86%(25)	82,10%(20)	82,92%(35)	74,44%(5)

Table 8.1: *Summary with the best results obtained*

By analyzing the results of the *Spambase* domain, it is noted that in the original data the results obtained with TL-KNN, TLN-KNN and S-KNN are slightly improved. However, as the number of random attributes increases, the improvement that is produced increases considerably. On the version *Spambase Rotated 8* and *Spambase Rotated 50*, we see how the reduction of the dimensionality that is done is very efficient, by getting to surpass 90% in the classification (with the linear transformation and the sigmoidal function) with a data set with only 2 attributes. It is important to note that the performance is improved in *Spambase Rotated 8* (65 attributes) and *Spambase Rotated 50* (107 attributes) with a new data set formed by only 2 attributes. Given this data, it is possible to conclude that in this domain it is highly profitable to perform the reduction of the dimensionality with the linear transformation or by applying the sigmoidal function.

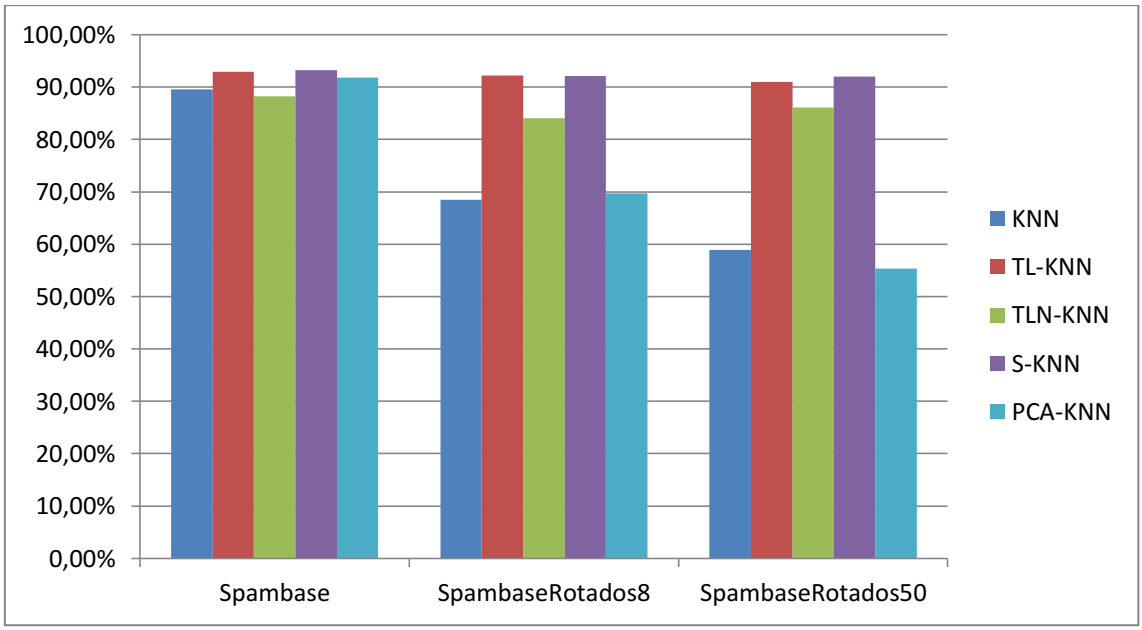

Figure 8.1: *Comparative chart in Spambase*

The tests on the domain known as *Doughnut* also have very good results with the reduction of the dimensionality carried out by the network of neurons in any of its three versions (TL, TLN or S). We can see that in both the *Doughnut 8* and the *Doughnut Rotated 8*, the results surpass the 99% of success (except in a single case that obtains

98.35%). In addition, it is important to highlight that using fewer attributes, it is obtained a result that is better than on the original data. The result obtained by PCA is also greatly improved. Again, it can be concluded that in this domain it is very profitable to use the reduction of dimensionality with the proposed model.

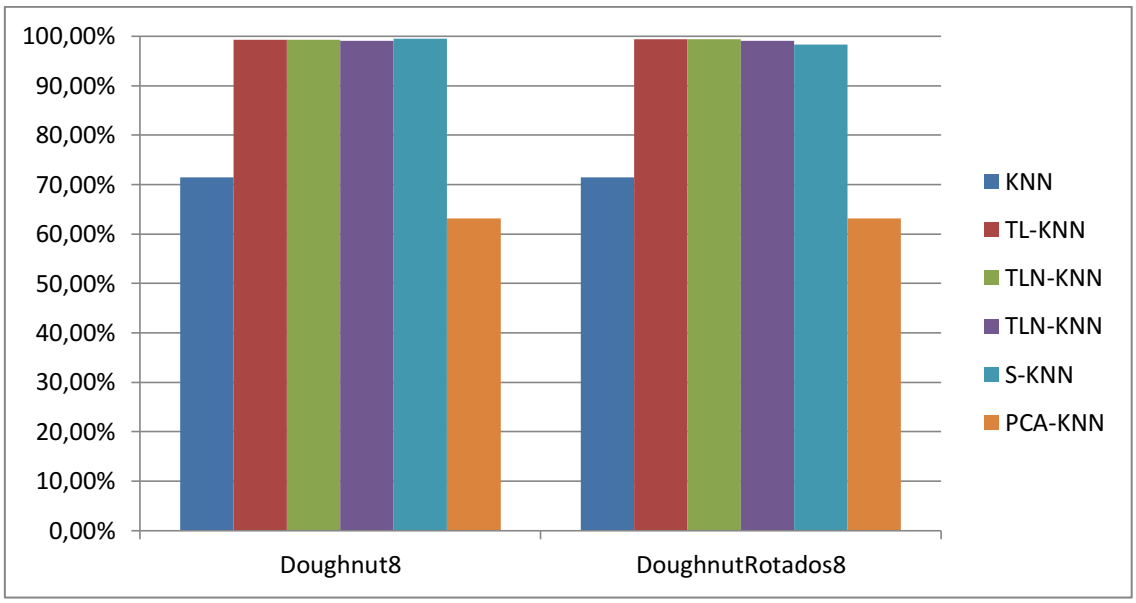

Figure 8.2: *Comparative chart in Doughnut*

However, in *German*, *German Rotated 8* and *German Rotated 24* there is not much benefit from reducing the dimensionality with the neural network in any of its three versions. *Table 8.1* shows a slight improvement in the success rate obtained on the original data (only in *German* with S-KNN the result is worse), but this improvement is not significant. Furthermore, it seems that the linear transformation in this case is a little more stable and offers a higher result than the normalized linear transformation and the sigmoidal transformation. Therefore, in this domain, although the improvement is not so high, it is also cost-effective to carry out the reduction of dimensionality by using the artificial neural network.

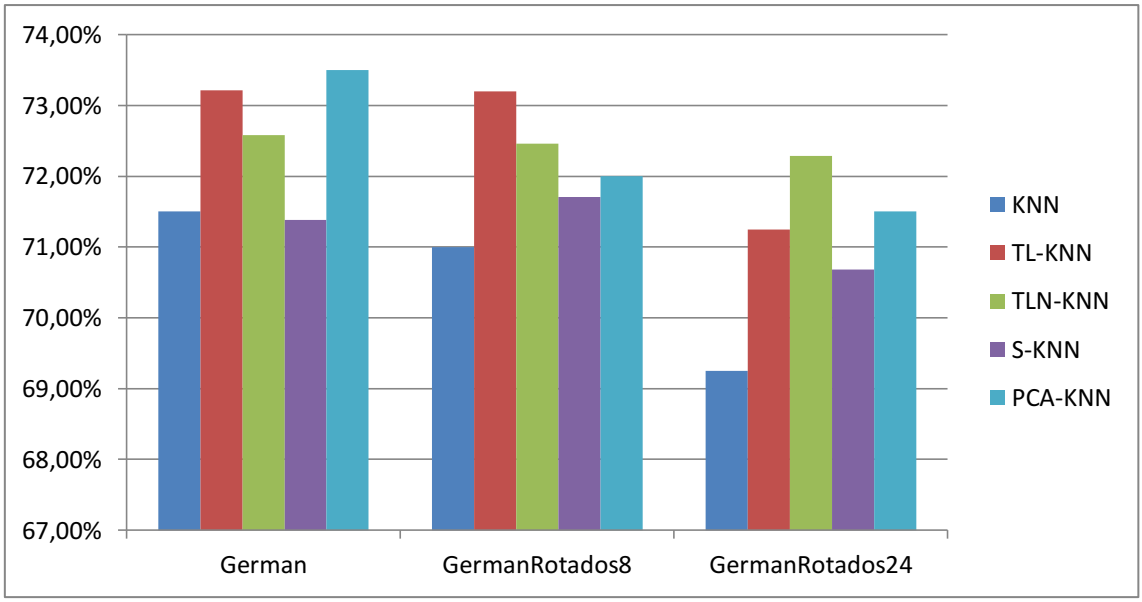

Figure 8.3: *Comparative chart in German*

Finally, it is observed in *Splice*, *Splice Rotated 8* and *Splice Rotated 20* as again it is advisable to do the reduction of the dimensionality by using the neural network, since the results obtained in the classification of the original data are improved. In addition, the linear transformation seems to be more stable in this case and offers better results than the normalized linear transformation and the sigmoidal transformation.

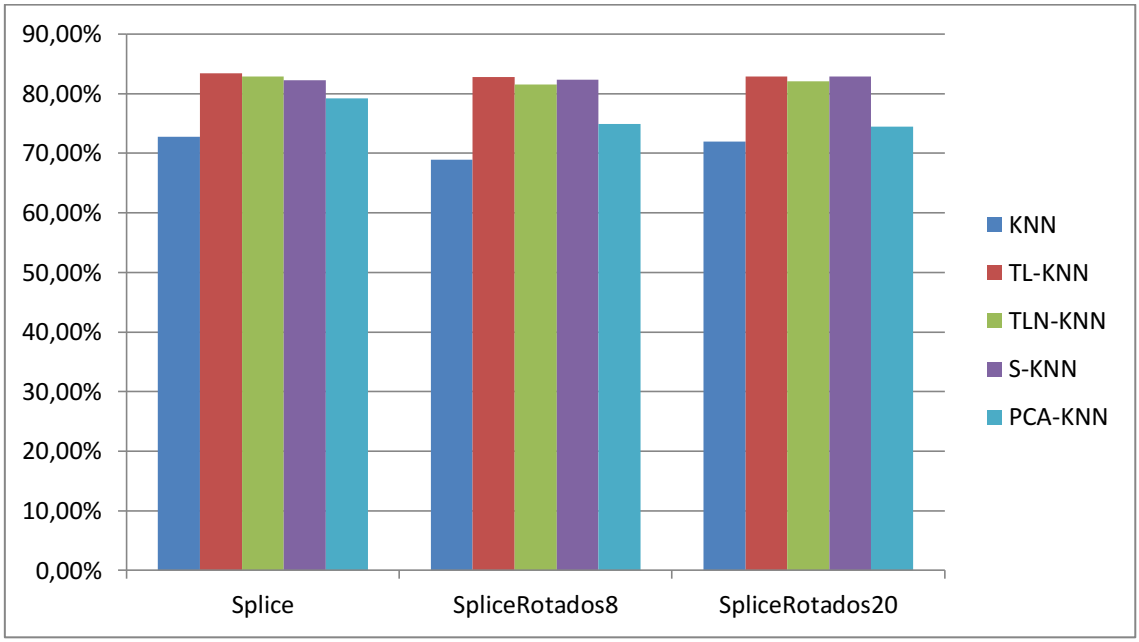

Figure 8.4: *Comparative chart in Splice*

Summary of the comparison of the reduction of dimensionality performed by the network of neurons and PCA

The following tables show the maximum value of improvement recorded in the comparison, as well as the average improvement obtained in each case. In the table negative values indicate a loss of precision with respect to PCA, on the contrary, if positive values are obtained, indicates that the results are improved.

Domain	Version	TL vs. PCA		TLN vs.PCA		S vs. PCA	
		Maximum	Mean	Maximum	Mean	Maximum	Mean
Spambase	Spambase	4,97%	1,69%	-2,94%	-9,17%	5,52%	1,86%
	SpambaseRotated8	38,35%	24,98%	28,12%	16,83%	40,53%	31,34%
	SpambaseRotated50	40,38%	31,25%	35,64%	31,25%	40,16%	32,76%
Doughnut	Doughnut8	44,61%	32,96%	44,61%	32,96%	37,15%	25,29%
	DoughnutRotated8	44,92%	33,30%	44,82%	33,18%	36,76%	25,81%
German	German	6,43%	1,84%	5,50%	1,17%	4,70%	5,13%
	GermanRotated8	8,29%	2,56%	8,63%	2,28%	8,46%	1,35%
	GermanRotated24	6,54%	3,07%	5,43%	3%	7,04%	2,66%
Splice	Splice	9,75%	5,71%	8,33%	4,43%	9,52%	4,56%
	SpliceRotated8	13,12%	9,03%	10,93%	8,09%	13,45%	8,41%
	SpliceRotated20	15,67%	10,07%	13,37%	9,29%	15,39%	9,12%

Table 8.2: *Summary of the comparison with the network trained with 100 iterations*

Domain	Version	TL vs. PCA		TLN vs.PCA		S vs. PCA	
		Maximum	Mean	Maximum	Mean	Maximum	Mean
Spambase	Spambase	5%	1,75%	-5,28%	-11,1%	5,61%	1,76%
	SpambaseRotated8	38,15%	25,03%	27,22%	15,08%	38,28%	26,45%
	SpambaseRotated50	40,53%	31,34%	31,28%	21,03%	40,61%	32,98%
Doughnut	Doughnut8	45,66%	34,45%	45,25%	34,52%	41,75%	28,21%
	DoughnutRotated8	43,53%	33,48%	45,35%	33,6%	41,22%	28,24%
German	German	5,92%	1,99%	6,50%	1,32%	5,63%	0,57%
	GermanRotated8	8,50%	2,96%	8,60%	2,15%	8,40%	2,14%
	GermanRotated24	6,40%	2,79%	6,50%	3,29%	7,80%	3%
Splice	Splice	10,47%	5,37%	8,56%	4,20%	9,63%	4,28%
	SpliceRotated8	13,46%	9,16%	12,17%	8,01%	14,21%	8,46%
	SpliceRotated20	15,11%	10%	12,63%	8,88%	15,75%	8,83%

Table 8.3: *Summary of the comparison with the network trained with 1.000 iterations*

In the previous tables, it is observed that only in the case of *Spambase* with the TLN a loss in the success rate is obtained, in the other cases positive results are obtained. It is also important to note that the improvement in many cases is very high, for example in some cases in the *Doughnut* domain the maximum is greater than 40% and the average is exceeded by more than 30%. Taking into account the data in the table, it can be concluded that the improvement that occurs with respect to PCA is very high.

As a conclusion to comment that any of the three versions studied (linear transformation, normalized linear transformation and sigmoidal transformation) seems to be viable as a dimensionality reduction technique. In the domains in which the study was carried out, the linear transformation has offered slightly better results in a generic way. In spite of this, it is not possible to recommend this method above the normalized linear transformation or the sigmoidal one and it is recommended to carry out a previous study with the three versions, choosing the one that gives the best results.

4. Conclusions

4.1. Goals fulfilled

In this document, chapter 2 describes existing dimensionality reduction techniques, as well as the problems presented by data sets with a large number of dimensions. In addition, this chapter presents similar works of studies carried out in the field of dimensionality reduction and some applications.

As detailed in Chapter 6, a script with the programming language R has been implemented to carry out dimensionality reduction experiments with the neural network, this script also includes a function that extracts the activations of the neurons of the hidden layer. Additionally, it is detailed how another script was created to apply KNN on the original data and to obtain the classification results of the reduction of the dimensionality performed using PCA. It should be noted the existence of functions in both script to perform a preprocessing of the data.

However, the main objective of the project was to study the feasibility of several dimensionality reduction techniques based on the Multilayer Perceptron, this is carried out empirically in chapter 7. In this chapter, the results of the experiments are showed,

as well as a comparison of the efficiency in the classification of the original data with the efficiency of the classification of the projections obtained by the neural network. In this chapter the comparison with PCA is also carried out.

The results of experimentation have been considered as good. Thus, a new model has been created for the library named Caret. This model performs a reduction of the dimensionality of the data before performing the classification task with KNN. The process of creating this model, as well as examples of its use are shown at the end of chapter 5.

4.2. Future lines of work

Because this is an experimental research paper, there have been some options that could not be explored because it had an excessive cost in time. Therefore, this section describes some of the possible future work that can be carried out.

A line that remains open is to perform the same study by using a classifier different from KNN to carry out the evaluation, since the results obtained by this algorithm are satisfactory, but other classification algorithms could obtain a negative result.

However, the main line of work that remains open is to study the feasibility of dimensionality reduction by using a neural network with more than one hidden layer. In this case, we would have to study the results obtained in the classification of the neuron activations of the last hidden layer, in order to carry out the evaluation KNN algorithm could also be used.